THE UNIVERSITY *of* EDINBURGH

Parallel computation in low-level vision

Andrew Blake

Ph.D Thesis
1983

Machine Intelligence Research Unit
University of Edinburgh

ABSTRACT

This thesis is concerned with problems of using computers to interpret scenes from television camera pictures. In particular, it tackles the problem of interpreting the picture in terms of lines and curves, rather like an artist's line drawing. This is very time consuming if done by a single, serial processor. However, if many processors were used simultaneously it could be done much more rapidly. In this thesis the task of line and curve extraction is expressed in terms of constraints, in a form that is susceptible to parallel computation. Iterative algorithms to perform this task have been designed and tested. They are proved to be convergent and to achieve the computation specified.

Some previous work on the design of properly convergent, parallel algorithms has drawn on the mathematics of optimisation by relaxation. This thesis develops the use of these techniques for applying "continuity constraints" in line and curve description. First, the constraints are imposed "almost everywhere" on the grey-tone picture data, in two dimensions. Some "discontinuities" - places where the constraints are not satisfied - remain, and they form the lines and curves required for picture interpretation Secondly, a similar process is applied along each line or curve to segment it. Discontinuities in the angle of the tangent along the line or curve mark the positions of vertices. In each case the process is executed in parallel throughout the picture. It is shown that the specification of such a process as an optimisation problem is non-convex and this means that an optimal solution cannot necessarily be found in a reasonable time A method is developed for efficiently achieving a good sub-optimal solution.

A parallel array processor is a large array of processor cells which can act simultaneously, throughout a picture. A software emulator of such a processor array was coded in C and a POP-2 based high level language, PARAPIC, to drive it was written and used to validate the parallel algorithms developed in the thesis

It is argued that the scope, in a vision system, of parallel methods such as those exploited in this work is extensive. The implications for the design of hardware to perform low-level vision are discussed and it is suggested that a machine consisting of fewer, more powerful cells than in a parallel array processor would execute the parallel algorithms more efficiently

Contents

Appendix

1 Introduction

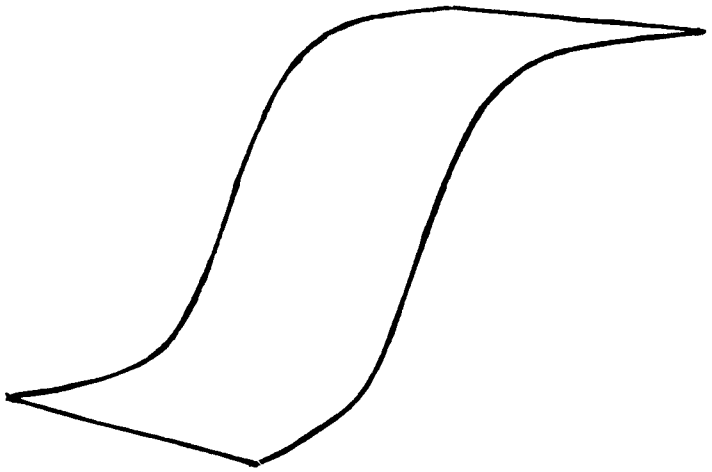
1.1 Cooperativity and relaxation

A cooperative process is a computation performed in parallel by a network of independent processing cells. Each cell is connected to just a few of its neighbours in the network, and continually performs a local computation - computing some function of its own current state, its own input signal, and the signals received from its neighbours. The computation continues until the network settles into a stable state.

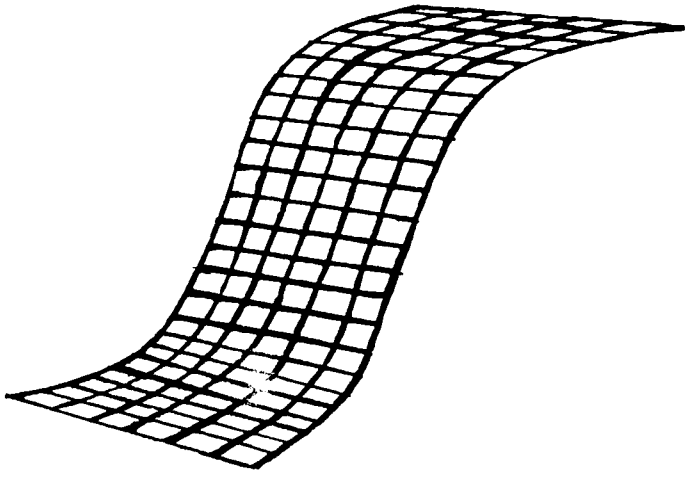
The set of inputs to the cells constitute the input to the network as a whole, and the final state of the network is the output of the computation. Clearly the final state is determined by the nature of the local computation performed by each cell, and by the pattern of inter-cellular connections. For such a scheme to be useful, it must meet two criteria: first there must be a guarantee that the network will not continue in activity indefinitely - it must come to rest in some stable state; secondly, having specified the desired output of the network for every possible input, the final stable state must deliver that output.

An example of a cooperative process is given in fig 1.1, in which an elastic sheet - a soap film for example - is to be stretched over a wire frame. The sheet takes up a minimum energy configuration as specified by Laplace's equation. In order to calculate what that configuration will be, the equation can be solved by relaxation, a discrete-time cooperative process. The shape taken up by the sheet is represented by its height at each point on some rectangular grid and initially some rough estimate of those heights is made. The relaxation process proceeds iteratively, repeatedly replacing the value of the height at each position inside the wire frame by the average of the values at the four neighbouring positions. While this goes on the heights of points on the wire frame itself remain fixed. The result is that the influence of the wire frame propagates inwards on the sheet and finally the sheet comes to rest in the minimum energy position.

In this example the network that was talked about earlier is the grid of points, carrying height values. There is a translational symmetry about this network because the same computation, namely averaging, takes place at each grid point inside the wire frame. Moreover the local averaging can be performed



a)



b)

Fig 1.1 a) A wire frame. b) An elastic sheet stretched over the frame The shape that the sheet assumes can be calculated iteratively and in parallel, by relaxation.

simultaneously - in parallel - at all grid points. These two properties, translational symmetry and parallelism, are characteristic of the use of relaxation in computer vision.

The motivation for investigating parallel processing in low level vision is strong. Builders of computer vision systems (e.g. Perkins (1978)) have often found low level vision, done on conventional serial computers, to be a processing bottleneck and the use of many processors in parallel could alleviate this. Studies of the mammalian brain strongly suggest that it uses parallelism to a high degree in vision (e.g. Hubel and Wiesel (1974b)), with relatively slow processing elements but many of them.

Feldman and Ballard (1982) have reviewed some current theory of parallel models of perception and in particular bring out an important issue concerning how quantities should be represented in a parallel network. They argue for the 'value unit', a boolean cell whose 'on' state signifies that a particular quantity has a given value. Thus to represent some numerical quantity a stack of value units is needed, one for every possible value that the quantity might have. Marr and Poggio (1976) used value units to compute stereo disparity and they are used in the brain to represent the local orientation of lines in the visual field (Hubel and Wiesel (1974a)). Constrained labelling - parallel network computations involving qualitative as opposed to numerical values - make natural use of value units (e.g. Waltz (1974)).

Value units have drawbacks for representing numerical data. If a quantity is to be represented with any precision a large number of units are required for each instance of the quantity; this is inefficient compared with a 'variable unit', either an analogue representation in which the quantity is proportional to some signal (e.g. electrical potential) or a digital representation in which a number in the range $[1..2^N]$ is represented using only N bits, compared with 2^N value units. Just one variable unit is sufficient to represent a numerical quantity, as compared with an entire stack of value units. The variable unit automatically incorporates the constraint that a quantity takes only one value at a time. (There are effective ways of using value units to represent numbers, like the Hough transform (Ballard and Brown (1982))). Variable units are found in the brain too - for instance the X cells in the retina and lateral geniculate body (Marr and Ullman (1981)) which can produce a graduated response to changes in intensity of light falling on the retina.

On digital computers, designed to perform arithmetic, variable units are the more convenient representation; arithmetic with value units can be somewhat clumsy, implemented as large logical networks - effectively as lookup tables. Network algorithms implemented using variable units could however be converted to use value units simply by expressing the local function computed by each network cell as a lookup table rather than as an arithmetic expression.

Data in low level vision initially appears in the form of regular grids or arrays. Intensity data from television cameras is usually arranged on a rectangular grid whilst hexagonal arrays of sensors are found in the eye (Andrews et al. (1973)). Therefore, initially at least, computation in vision must deal with the data in array format.

Barrow and Tenenbaum (1978) proposed the "intrinsic images" model of cooperative computation for low level vision In this model relaxation processes, rather like the surface fitting algorithm already described, are used to maintain the continuity and mutual consistency of various array quantities that describe a 3-D scene· intensity, surface orientation, reflectance, range etc.. Examples of the sort of relaxation algorithm that might be appropriate are given by Ikeuchi and Horn (1980), Horn and Schunk (1980), Barrow and Tenenbaum (1981) and Narayanan et al. (1982) Each of these algorithms fit various kinds of data to smooth functions. The principal contribution of this thesis is to develop this idea to fit data to *piecewise* smooth functions - surfaces that are smooth almost everywhere but can also have some breaks and creases, termed *discontinuities*. Discontinuities found in the quantities that describe a scene form lines and curves which can be used as basic primitives or features for object recognition and scene description.

1.2 Continuity constraint and optimisation.

An important task in low level vision is the labelling of discontinuities (often called edges) in the intrinsic images (Marr (1982), Barrow and Tenenbaum (1978), Binford (1981)). The simplest approach to locating discontinuities in discrete arrays is to use a local differentiation operator (for examples see Ballard and Brown (1982)). These are high pass filters that act over some small neighbourhood of each pixel†. Peaks in the output of the operator are taken to

† A pixel or picture element is a single element in an image array

be the locations of edges, but with real, noisy data this gives spurious responses in some places, and missed responses in others (gaps in edges). Suppression of response to noise can be achieved by reference to the context of each pixel within the array, for instance by an operator with a large neighbourhood such as the difference of gaussians (or DOG) (Marr and Ullman (1979)). Noisy responses are suppressed but significant errors in the positions of edges can occur (see chapter 2). Alternatively edge tracking (e.g. Nevatia and Babu (1979) and region analysis (Brice and Fennema (1970)) make some reference to context without using operators with a large, fixed (data-independent) neighbourhood. Neither, however lend themselves easily to cooperative implementation

In this thesis the task of labelling discontinuities in a certain array quantity (e.g an array of intensity values or any intrinsic image) is formulated as the satisfaction of a certain constraint: that the quantity is continuous almost everywhere (see Marr (1978)). The constraint is applied by a relaxation algorithm in which reference to context can occur throughout the array, not limited to some fixed neighbourhood. The algorithm is a best-fitting process that proceeds simultaneously throughout the array, fitting the initial data array to a distribution that is continuous almost everywhere. The contour of a discontinuity in the final distribution may have arbitrary shape - there are no restrictions of local linearity (as there are, for instance, in Hueckel's (1971) large neighbourhood operator). However almost exactly the same relaxation process can subsequently be applied to finding straight lines and vertices between them.

The task is defined as an optimisation problem that positions discontinuities optimally and at the same time constructs smooth surfaces within boundaries defined by the discontinuities. Given an initial array of data, a new array is constructed which minimises the following measure of cost:

the sum of squares distance of the new array from the initial one

PLUS

the sum of penalties exacted in return for allowing breaks between smooth regions. One penalty is paid for every unit of length of the common boundary between two regions.

Note that an edge-region duality constraint is implicit here: edges are discontinuities between continuous regions in the new array and hence are automatically continuous themselves (in the sense that they are unbroken).

This thesis deals mostly with a rather restricted interpretation of continuity as flatness or "constancy" so that the result of relaxation is a piecewise constant array - constant regions separated by breaks. This is an over-simplification of

the real world; generalisation to cope with sloped and curved functions is considered in chapter 2.

Finding a global minimum of the cost function described previously proves to be an intractable task in general, because the function has the mathematical property of non-convexity, a stumbling block to standard techniques for optimisation. This property occurs as a consequence of the sudden way in which, in the cost function, a penalty is applied as soon as even the smallest discontinuity appears. A method is proposed ("convex envelopes" and "graduated non-convexity", in chapter 4) that enables a good suboptimal solution to be achieved.

The method consists of a series of optimisations performed on each of a particular sequence of cost functions, ending up with the cost function described above. An appropriate physical model of the algorithm uses similar apparatus to the illustration given by Julesz (1971) of his cooperative stereo algorithm. Bar magnets on springs are arranged in a regular array and interact with one another in a way that tends to align them with their neighbours (see fig 1.2). The analogy here is between numbers in an image array and the angles of orientation of the magnets. The springs on the magnets tend to return them to their initial positions. The force of interaction between magnets is made to change gradually with time (an analogy for the sequence of cost functions). At first the force is weak, gradually becoming strong and short-range - pulling hard to align adjacent magnets but only if they are nearly aligned already.

An example of the results of this algorithm used on image intensities is shown in fig 1.3. The effect of using different sizes of penalty for allowing breaks between regions can be clearly seen: a large penalty discourages the formation of discontinuities and produces a very sparse line drawing, showing only the most persistent discontinuities.

Finally, if an array of numbers is used to represent local orientation - that is, the angle of the tangent to lines in the line drawing that has been obtained already by relaxation - the same relaxation techniques can be used on that array too, to impose weak constancy constraints on local orientation. An initial array of angles can be obtained from the image itself by using a suitable local operator (e.g. O'Gorman (1978)). (The situation now corresponds almost exactly to the bar magnet model: magnet angles are now an analogy for local angles of line segments.) The relaxation process is restricted to take place only along the discontinuities of intensity found by the previous relaxation process. The result is an array of piecewise constant orientation in which angle discontinuities mark vertices between straight line segments. An example is given in fig 1.4.

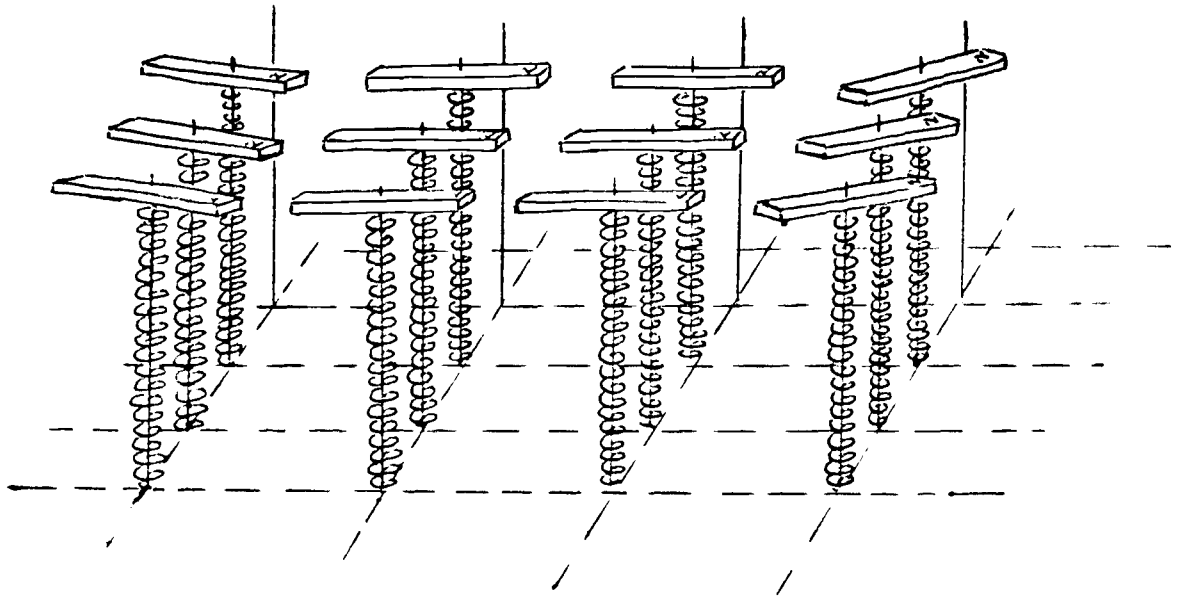


Fig 1.2 A useful physical analogy for weak constancy is a regular array of magnets on springs. The springs tend to return the magnets to their 'natural' position, but the forces between magnets tend to align them with their neighbours. (Note that the magnets are mounted on pivots, free to rotate in the horizontal plane; they have no other degrees of freedom. Therefore, in this analogy it is the angles of the magnets in the horizontal plane that are subject to weak constancy constraints.)

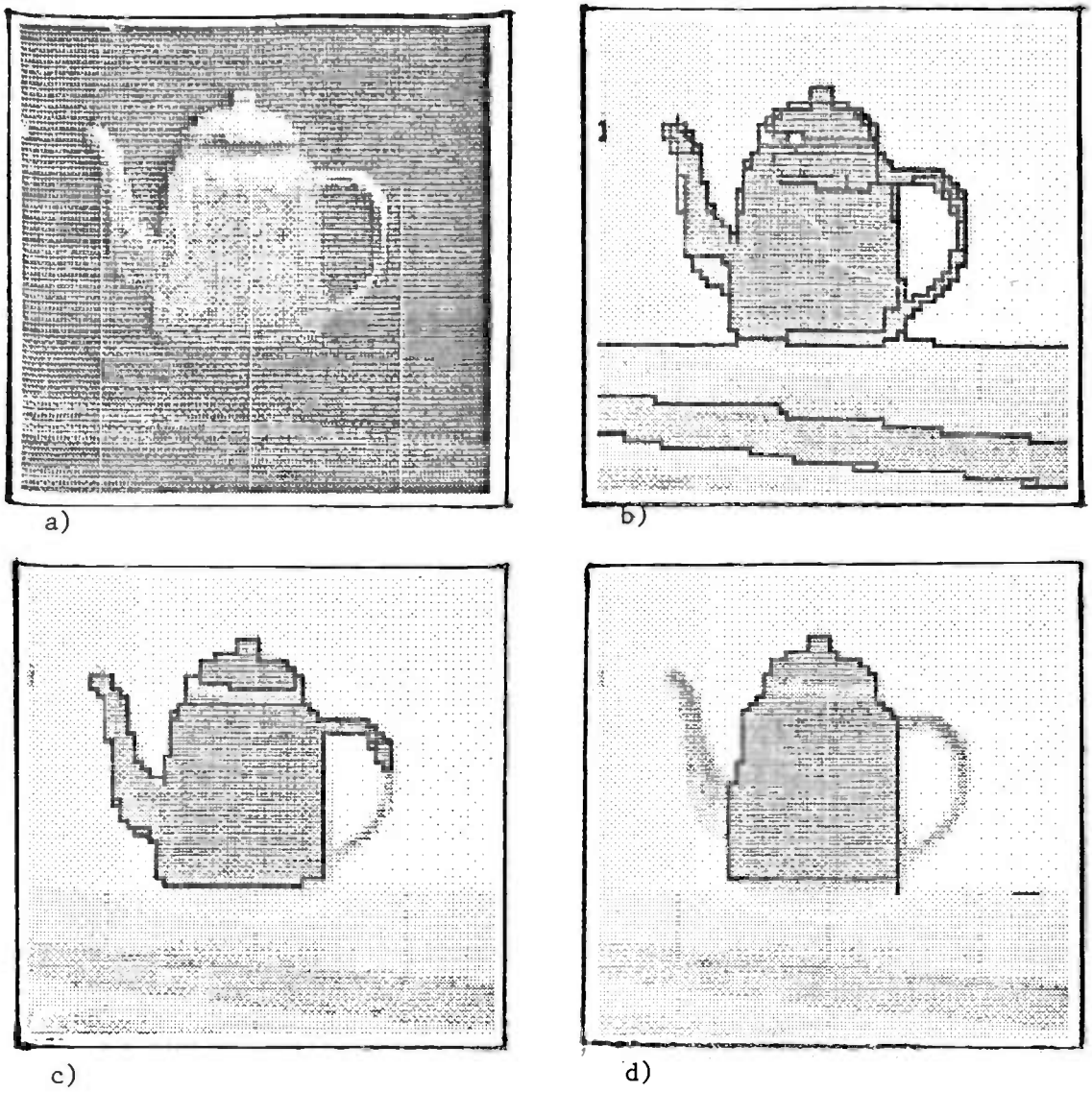
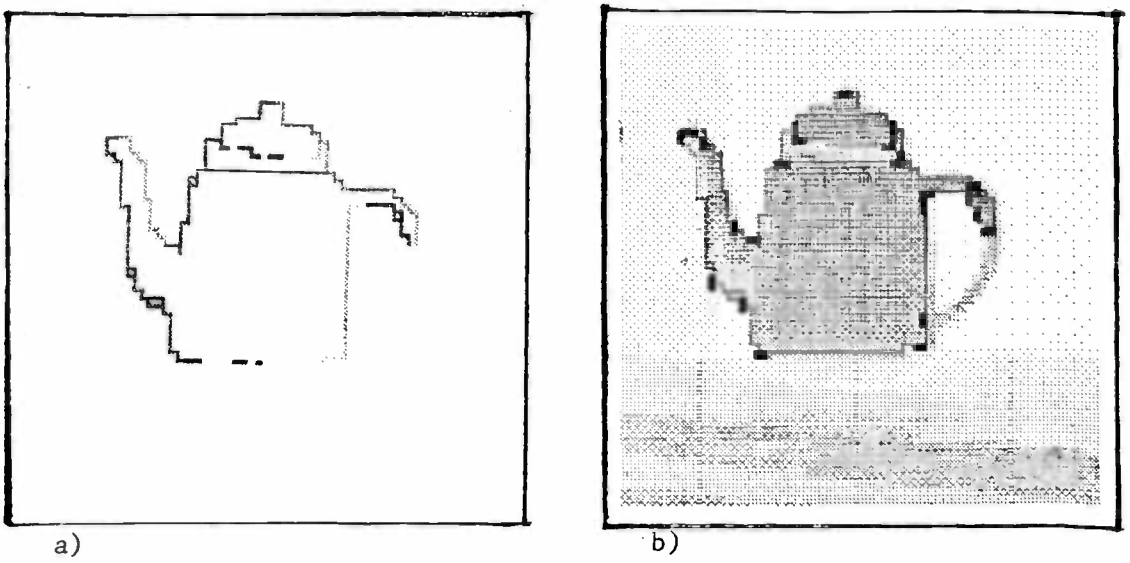


Fig 1.3 The weak constancy relaxation algorithm applied to a) produces the results in b), c) and d). Increasing values of the penalty constant (see text) produces successively sparser line drawings. This is because the penalty is a measure of reluctance to allow a line. (Actual values of penalty are C=64,128,256 respectively.)



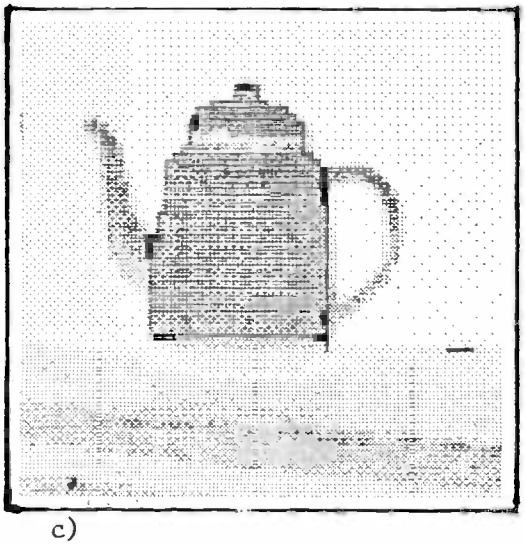


Fig 1.4 A suitable local operator, applied to fig 1.3a, produces the angle array in a): 0 to 360° is represented by brightness, from dark to light. Weak constancy relaxation can be applied to this angle data to find straight line segments in a line drawing, and mark the vertices between them. Vertices are shown in b,c, marked by dots, (b,c correspond to fig 1.3 c,d).

1.3 Parallel hardware

The motivation to develop the parallel algorithms described in this thesis arose initially from the existence of parallel array processors like the CLIP4 (Duff (1978)). It is an array of nearly 10000 simple processing cells which perform operations on boolean arrays - arrays of single bit truth values. Boolean arrays can be stacked to represent integer arrays and boolean operations can be combined to perform integer arithmetic in a highly parallel manner.

However, although such a machine provides a great deal of raw computing power (about 60 16-bit additions per microsecond) the advantage that this brings to relaxation algorithms turns out to be limited. The limitations are threefold:

- the cells can only operate synchronously, all doing exactly the same thing at the same time
- memory space at each cell is very limited
- in sparse arrays cells in empty areas of the array lie idle

Relaxation algorithms of the sort described in this thesis could make much more effective use of a machine with an asynchronous architecture. This would consist of a number (10-100 perhaps) of powerful arithmetic processors, each with plenty of local memory, each independently executing its own stored program, and each with independent random access to data arrays. Given processors with the ability to perform a 16-bit addition in a microsecond, just 8 of them, working asynchronously, would be sufficient to perform weak constancy relaxation at a rate comparable with CLIP4. For faster performance yet, 20 processors are enough to complete the algorithm, on 64×64 images, in about a second. There is still scope for further tuning of the implementation of the algorithm itself, to obtain even greater execution speed.

2 Expressing processes in low-level vision as constrained labelling

The idea of affixing labels to nodes in a graph, under certain constraints, has been widely discussed, both in computer vision and other applications. A general reference on this subject is Davis and Rosenfeld (1981). In a constrained labelling scheme, the way in which any particular node in the graph may be labelled depends on the labels on certain of the other nodes. The arcs of the graph join precisely those nodes whose labels are mutually constrained in this way. A labelling of the graph which satisfies all the constraints is termed a "consistent" labelling.

The constrained optimisation formalism is an elegant way of expressing certain kinds of computational process. The topological structure of the graph defines the data structure over which the computation is to take place and the constraints themselves constitute a declarative specification of the process; the goal state is that the constraints should be satisfied but the details of how this is to be achieved (the labelling algorithm) are not contained in the process definition. The specification is non-deterministic in the sense that there may be more than one consistent labelling, any of which would be acceptable as the final state of the process.

While chapters 3, 4 and 5 deal with the design of algorithms for obtaining consistent labellings, this chapter concentrates on the way in which some processes in vision can be specified in terms of constraints. The first section (2.1) discusses briefly some current ideas in the literature on the overall structure of a vision system. In later sections some tasks within that structure are specified in terms of imposing continuity constraints on labels in a graph. In these cases the graph is a highly regular data structure - an image array - and the labels are real valued quantities, namely image intensity and line segment orientation.

In section 2.3 a continuity constraint (the edge-region duality constraint) is derived for the strengths of a chain of edges. (Edges mark the positions of discontinuities in an array of values and their strengths measure the size of the discontinuity.) It transpires that this is of limited use because it is not sufficient, on its own, fully to specify the process of describing discontinuities in an array.

A more general and powerful notion is the "weak" constraint, a constraint that holds almost (but not entirely) everywhere. Continuity almost everywhere is crucial in low-level vision. Quantities like image intensity (Binford (1981)), surface orientation (Ikeuchi and Horn (1980)), stereo disparity (Marr (1978)) and optical flow (Horn and Schunk (1980)) all obey that weak constraint. This thesis proposes the *weak continuity constraint* as a precise expression of "continuity almost everywhere". Moreover, in chapter 4, an algorithm is derived which performs a labelling process under weak continuity constraints. The algorithm works by relaxation. It applies local operations to an array, iteratively and in parallel, to achieve a globally consistent labelling. It is proved to be convergent - it always terminates.

2.1 Models of low-level visual processing

An important issue in the design of a vision system is how far low-level visual processing can be regarded as independent of high-level, interpretative processes. What is the scope of prior knowledge and reasoning in the perception of a visual scene? Is it confined to three dimensional interpretation or does it extend to perception of image primitives such as lines. For instance the Kanisza's triangle illusion (fig 2.1) could be caused by low-level edge detection machinery, or by an attempt at a higher level to deduce the presence of an occluding triangular shape (Frisby (1979)). Marr (1978) proposes the knowledge-free formation of a "primal sketch" which marks discontinuities of intensity in the image. This is used with stereo matching to construct a 2½-D sketch, which is a viewer centred description of the position and orientation of surfaces in the scene. At this stage the only knowledge used has been a set of general, fixed constraints about properties of 3-D surfaces. Only after this has been done is scene specific knowledge invoked to infer the 3-D structure of the scene. On the other hand, Feldman and Yakimovsky (1974) describe a way in which region analysis (e.g. Brice and Fennema (1970)) can be augmented by interpretation of the type of each region. Region analysis is essentially the dual process to edge analysis and therefore at a comparable level with the primal sketch. As an example of how interpretation can help, consider classifying regions in an outdoor scene as grass, sea or sky. Sky is usually found at the top of an image, and above grass and sea. This constraint can be used to help

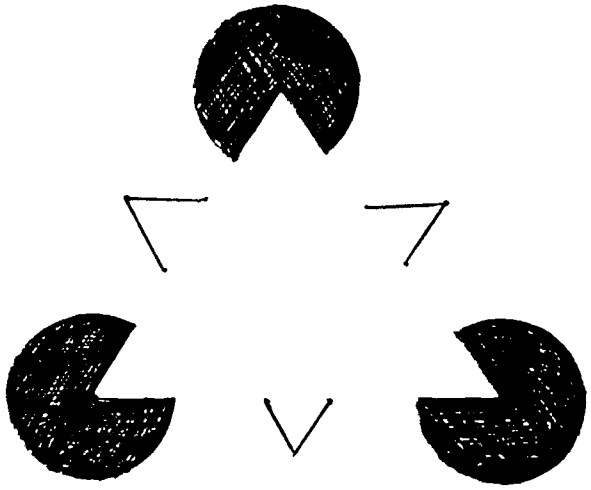


Fig 2.1 The Kanisza's triangle illusion. Illusory contours can be seen around a triangular shape in the centre.

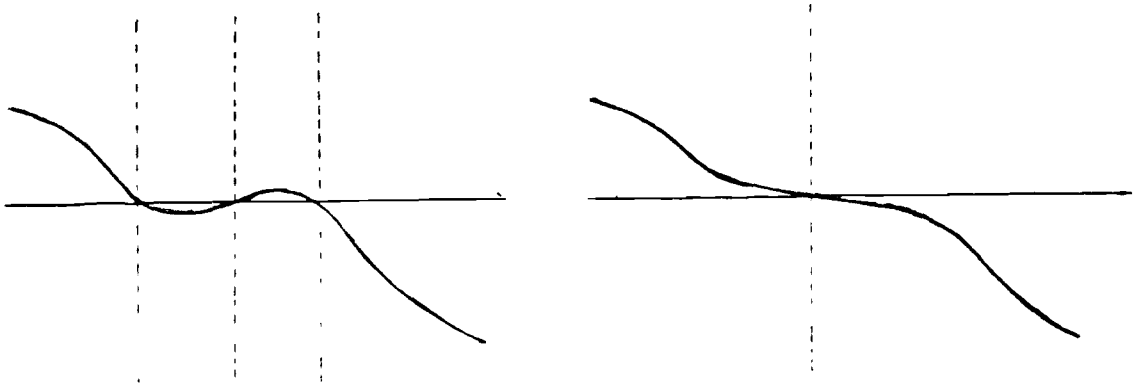


Fig 2.2 Two very similar functions with entirely different zero crossings.

identify regions as sky. Two adjacent regions identified as sky can then be merged because of their common interpretation. Similarly Shirai (1975) used a line finder assisted by knowledge about the sort of objects populating the 3-D polyhedral world in which it operated.

Of computer vision systems that have been built and described in the literature, Ambler et al. (1975), Perkins (1978) and Brooks et al. (1979) all use self-contained, knowledge-free processes at the level of line and region finding. Hanson and Riseman (1978) similarly use knowledge-free processing to construct their region-segment-vertex graph but do allow for the possibility of later semantic refinement.

The "intrinsic image" model of low-level visual processing (Barrow and Tenenbaum (1978)) can also be knowledge-free but allows for the influence of high-level knowledge. An intrinsic image is an array structure containing information about a measured physical quantity (e.g. image intensity) or an inferred quantity (e.g surface orientation, surface reflectance). Barrow and Tenenbaum propose the use of relaxation to achieve continuity within each image and consistency between images (in the sense of adherence to the physical laws of lighting, imaging etc.). The "stack" of intrinsic images forms a structure which does not necessarily require knowledge to construct its low-level description. Neither is the use of knowledge precluded, in that high-level knowledge may force an interpretation by imposing extra constraints (e.g. a priori knowledge of the orientation of the surface of the sea) or provide missing physical data (e.g. knowledge of likely ambient lighting conditions). The relaxation mechanisms, proposed in this thesis, for constrained labelling are well suited to handling intrinsic images. This should become apparent, especially later in this chapter, and in chapter 4.

2.1.1 The primal and 2 1/2-D sketches

Marr's primal sketch is composed of tokens which mark points of special interest in the image, such as discontinuities in intensity (lines and bars). These tokens mark important points in the scene itself, such as object boundaries and surface marks, and should therefore display some invariance under changing viewing conditions (illumination, orientation, perspective). Given that invariance they are suitable for use in determining correspondence between two stereo images: the two eyes view under different conditions but it should still be possible to pair corresponding tokens in the two images, because of their invariance. When the correspondence between the two eyes view has been determined the

result is used to construct the 2½-D sketch, in which stereo disparity is presented as a description of surfaces in the scene, in viewer's coordinates. Interaction between low-level and high-level occurs here in the 2½-D sketch, which serves as a buffer, written by low-level processes and read by 3-D interpretative processes.

2.1.2 Zero-crossings

Marr (1982) argues for the use of a linear $\nabla^2 G$ filter to extract tokens for the primal sketch. It consists of a gaussian operator followed by a laplacian. The zero crossings of the result (the points at which this result function changes sign) mark the positions of the edge tokens. This operator is approximately the same as the DOG (difference of gaussians) that has been used in psychophysical models of the human visual system (Wilson and Bergen (1979)) in which there are a number of filter channels each based on gaussians of different sizes.

Marr argues for the use of zero crossings of such channels on theoretical grounds (Marr and Ullman (1979)). Logan's theorem (Logan (1977)), a sampling theorem, is invoked: that under certain conditions a function is determined entirely by its zero crossings (up to a multiplicative constant). The important condition is that the function be bandlimited within one octave. The $\nabla^2 G$ filter, however, does not have a sharp cutoff in its frequency response, but the response is mostly within 1½ octaves. Marr claims that Logan's theorem is still of relevance in this case, the zeros can be regarded as "rich in information", a good, though not perfect, characterisation of the function. However this line of reasoning is inadmissible: Logan makes it clear that his theorem has no validity whatever when the bandwidth exceeds one octave. It might be argued that a function f_1 which is band limited almost within an octave could be closely approximated by a function f_2 whose spectrum lies entirely within an octave. This argument becomes invalid when extended to the *zeros* of the functions. This is because two very similar functions can have entirely dissimilar zeros as, for example, in fig 2.2. The zeros of f_1 may therefore be entirely unrelated to those of its octave band limited approximation f_2. It cannot be said therefore that the zeros of f_1 approximately represent f_1.

Logan also points out that although a function may be determined by its zero crossings, there is no readily available transformation from a set of zeros back to the function. In any case the intention was to represent not the function itself but "scene invariants" such as discontinuities in intensity. For instance the zero of a $\nabla^2 G$ operator correctly marks the position of an isolated step

discontinuity or peak in gradient as in fig 2.3. But in other cases an error in position can arise - when there is asymmetry (fig 2.4b) or an interaction between edges (fig 2.4d) - an error whose magnitude is of the same order as the width of the $\nabla^2 G$ pointspread function (fig 2.3e). Therefore when a $\nabla^2 G$ operator with a large pointspread function is used as a coarse operator to locate major edges but exclude small detail, edges so found may have substantial positional errors; an edge may be displaced relative to the image discontinuity that generated it. These errors could, in principle, adversely affect stereo matching because the size of the displacement may be different in each eye. This is because the size of the displacement is partly dependent on quantities (e.g. intensity gradient) which are not invariant under a change of viewing conditions. However the effect is to produce a "second order" error in stereo disparity (the *difference* between the displacements in each eye) and may therefore be insignificant. The displacements are more likely to be significant in object description as they can cause substantial geometrical distortion. This is a "first order" error.

Binford (1981) argues against the use in edge detection of large linear operators, for much the same reason. Instead he advocates sequential boundary tracking which accurately traces the path of intensity discontinuities. This thesis proposes a way of combining the advantages of both methods: susceptibility to parallel computation (linear filters have this property) and accuracy in locating intensity discontinuities The result is a non-linear filter, whose design clearly and explicitly reflects the semantics of labelling discontinuities.

2.1.3 Intrinsic images

Intrinsic images (Barrow and Tenenbaum (1978)) constitute a data structure in which to generate a consistent physical description of a scene. They are best visualised as a stack of arrays each of which represents one physical quantity in the scene, as a scalar or vector field. In a vision system, processing of data in array format can take place in a stack of intrinsic images and can also interact with high-level symbolic processing.

Processes in the image stack enforce *intra-image* constraints that must be satisfied within each image array, and *inter-image* constraints that act on two or more arrays. The latter reflect physical laws, such as the dependence of intensity on illumination, surface orientation and reflectance, or representational relationships as between range and its spatial derivative, surface orientation. The relationship between range and surface orientation is linear, just as with edge-region duality (chapter 2 3). The methods described in chapter 4 for linear

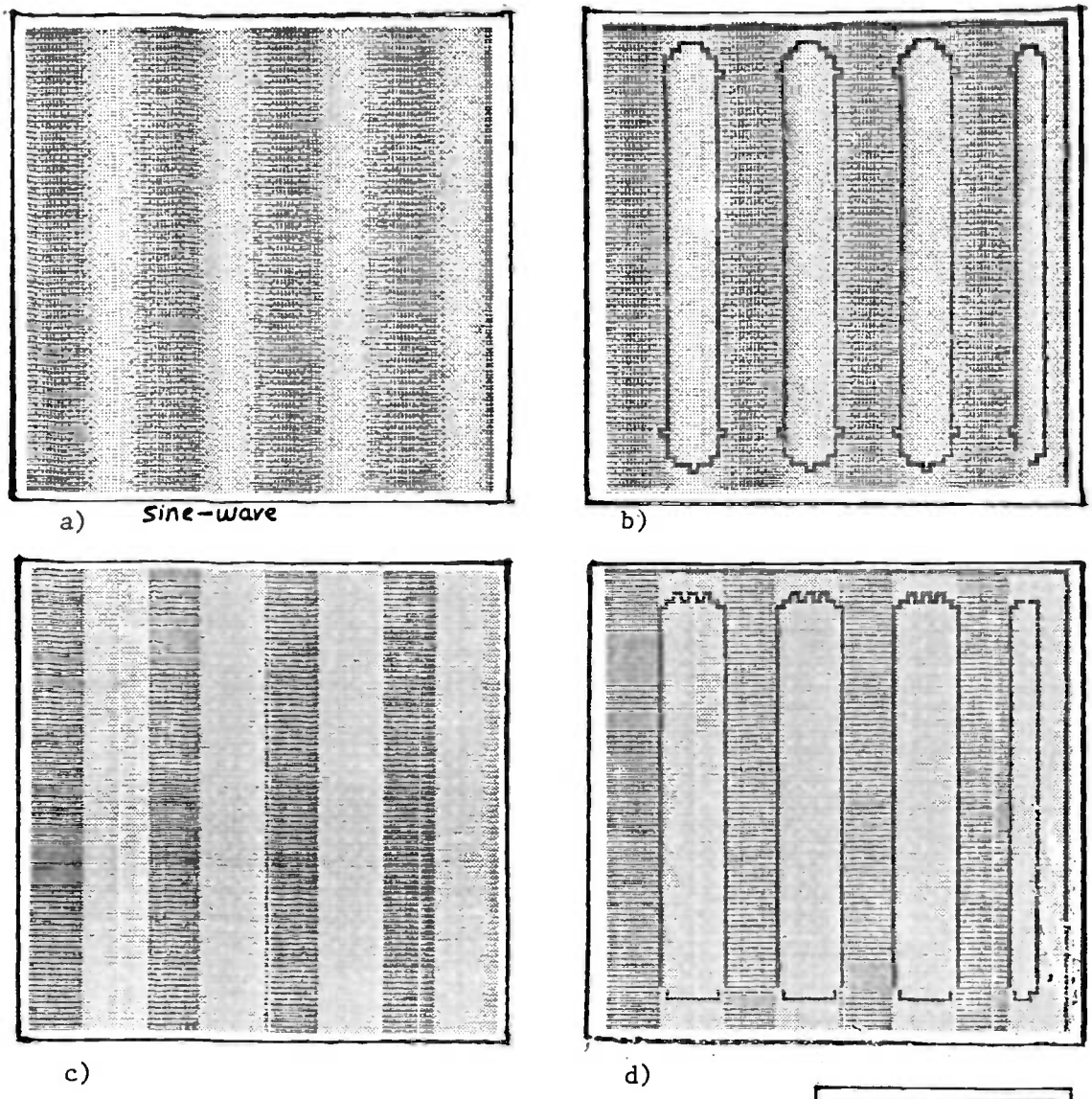


Fig 2.3 The zeros of a $\nabla^2 G$ operator applied to a & c mark discontinuities correctly, as shown in b & d. The pointspread function of the operator is shown in e.

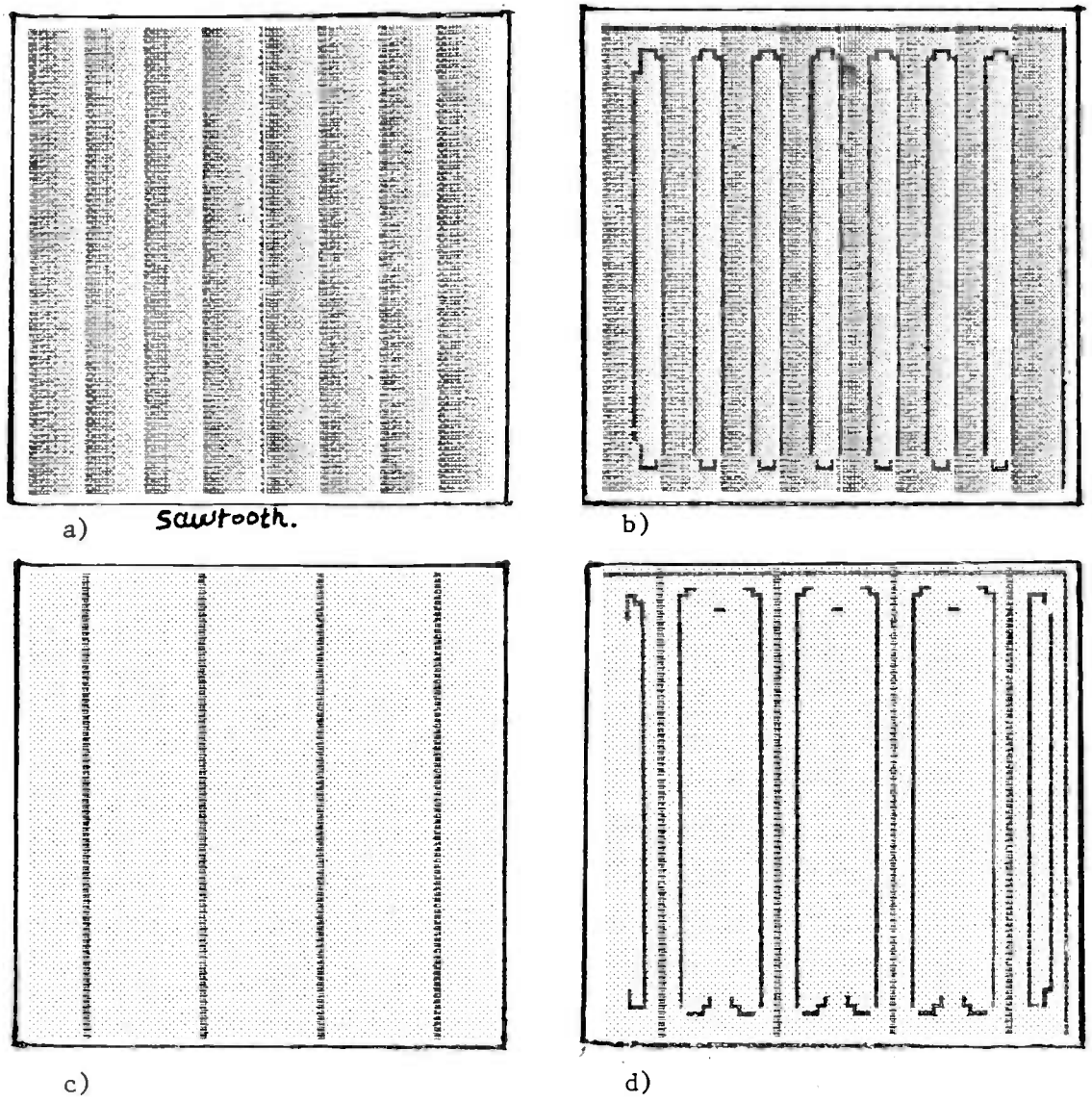


Fig 2.4 The zeros of $\nabla^2 G$ operator can produce erroneous markings either when the distribution of the array data is asymmetric (a and its zeros, b) or when two nearby edges interact (c,d).

constraints could also be used to maintain these relationships.

Within each intrinsic image continuity constraints ensure that the quantity represented by the image is continuous "almost everywhere". It has been suggested that this should be done by maximising some smoothness measure within predetermined boundaries. For instance Barrow and Tenenbaum assume that quantisation and noise are not a problem so that step changes in the intensity image can easily be located. Continuity is then imposed within boundaries defined by the step edges, by iterative local averaging. Similarly for surface orientation Ikeuchi and Horn (1980) require predetermined boundaries, within which surface orientation can be assumed to be smooth. But purely local operations (inspection of the data array in small neighbourhoods of each point) are inadequate in real images. Local edge operators tend to make either an overgenerous classification of discontinuities or a patchy classification, interrupted by noise (Ballard and Brown (1982) p. 83 ff.). On the other hand, larger neighbourhood operators suffer from inaccuracy, as with the $\nabla^2 G$ operator discussed earlier. The conclusion is that the classification of discontinuities is an essentially global process, and this is discussed further in section 2.2.

The view taken in this thesis is that making decisions on the labelling of discontinuities and imposing continuity within those boundaries should be done as a single integrated process. The process is defined here as the imposition of weak continuity constraints that the array quantity is continuous everywhere. The process itself produces an array (that is continuous almost everywhere) rather than a set of discontinuity markers. But, once that array has been obtained, local differentiation and thresholding suffice to extract explicit tokens from it; this can be done very rapidly compared with the task of imposing the constraints. Even after obtaining the discontinuity tokens further inspection of the array near the discontinuity may be desirable - for instance, in an intensity image, to classify the edge as shadow edge, occluding etc (Barrow and Tenenbaum (1981), Witkin (1982)).

The intrinsic image stack can be a cooperative process as well as a data structure. Inter-image constraints and intra-image (weak continuity) constraints are imposed simultaneously. Processes act in parallel throughout the stack to enforce the constraints and also take account of *a priori* information. This could either be sensed data such as intensity and possibly range, or high-level knowledge about ambient illumination etc. or about current provisional scene hypotheses (which might, say, fix the orientation of some surface). In a real time vision system the processes would run continuously to maintain

consistency in the stack by changing images in response to changes in *a priori* information. This arrangement automatically copes with occasions on which certain *a priori* information may cease to be available. For instance laser ranger data would substantially (subject to weak continuity and consistency with other images) determine the range image, but if it breaks down the range image is determined only by the intra- and inter-image constraints. In that way the best use is made of whatever information is available to fill in gaps in information with reasonable, consistent estimates.

Barrow and Tenenbaum's scheme, in which discontinuities fixed by examining the intensity image are accepted by other images, is hierarchical Information flows from the intensity image to the other images. What is proposed here is that each image should have equal status. Inter-image constraints are undirected arcs: information can flow in either direction between the members of a pair of mutually constrained images. This is in the same spirit as the BRPS conjecture of Mayhew and Frisby (1981), that primal sketch formation and computation of stereo disparity occur as one joint process. The conjecture is distinguished from the alternative hierarchical view of Marr and Poggio (1979) that the primal sketch is formed first and its discontinuity tokens in each binocular channel are subsequently matched to obtain disparities. Expressed in terms of intrinsic images the conjecture means that the intensity image and the range image are connected by a two way link. The formation of each is assisted by the other.

The lack of a hierarchy amongst intrinsic images means that processes on each image can run wholly asynchronously and in parallel - no process need wait for another to complete, before starting. The control structure is therefore simple - all inter-process communication is via input/output - and allows maximum parallelism in computation.

2.1.4 Lines and curves

Several computer vision systems have used lines and curves as primitives for object description. For instance the Brice and Fennema (1970) region analyser, after determining region boundaries, fits a bar mask in successive positions along a boundary to find discontinuities This partitions the boundaries into continuous segments which then serve as primitives for object analysis and recognition. Perkins (1978) in his system for part recognition assembles "concurves" (continuous curves) by partitioning object boundaries at curvature maxima. Concurves are used as primitives for matching an object to a model. Brooks

et al. (1979) in the ACRONYM system use the Nevatia and Babu (1979) line finder to form a line description of the scene. It is an angle sensitive boundary tracker that maintains continuity of orientation along each boundary. Hanson and Riseman (1978), in their VISIONS system, use cubic splines to fit smooth curves to boundary segments.

Each of those systems fit boundaries to a "model" in which boundaries are continuous except at vertices. Over an image the boundaries are continuous "almost everywhere" - just the same constraint as applies within an intrinsic image. This suggests representing boundaries by an "angle" intrinsic image, which contains an estimate of the tangent to lines of discontinuity in the intensity image. Where the intensity is continuous, elements of the angle image have no meaning. Along lines of *intensity discontinuity* weak continuity of angle is maintained, and *angle discontinuities* mark the positions of vertices.

Montavo and Weisstein (1979) give some psychophysical evidence (suggestive but not conclusive) that human perception of oriented line segments is a cooperative process. They base this on observed lateral inhibition in the detection by human subjects of pairs of adjacent line segments. However this could also be explained simply by crosstalk in spatially adjacent, orientationally tuned channels. Zucker et al. (1977) and Davis et al. (1977) also discuss the use of cooperative processes for organising chains of oriented line segments into lines and curves. The specification of this task in terms of weak continuity leads to a sound design for a relaxation algorithm to achieve the segmentation This is discussed further in chapter 4.

2.2 Continuity

From the discussion in this chapter so far we have seen that the idea of continuity almost everywhere is of fundamental importance to low-level vision processing. In a stack of intrinsic images cooperative processes fit each image to a piecewise continuous model. We need more precise definitions of "continuity" and "almost everywhere" in order to be able to specify the cooperative algorithms.

2.2.1 Continuity in a digital array

The difficulty in defining continuity in an array I_{ij}, $i,j=1..n$ arises from the process of sampling. The array I is a discrete representation of an underlying function $f(x,y), 0 \le x \le n, 0 \le y \le n$. The sampling process defines I in terms of f:

$$\forall i,j \; I_{ij} = \int_{i-1}^{i} \int_{j-1}^{j} f(x,y)\,dx\,dy. \qquad (2.1)$$

Much of the information in f is lost in the integration - this loss of information is quantified by the sampling theorem (Rabiner and Gold (1975)). In particular, continuity of the function f is well defined in classical mathematical analysis theory. But sampling obscures the issue because in the sampled array I, a discontinuity in f cannot be distinguished from a steep gradient in f. Looked at from the point of view of fourier analysis this phenomenon is known as aliasing: information is lost if the function f contains signal at higher than half the sampling frequency.

Additional uncertainty arises from quantisation of I, when each integer I_{ij} is represented in digital form as, say, an 8 bit word. It is impossible, therefore, reliably to detect small changes in the function. A difference in quantised value, between adjacent elements of I, of one grey level* can be caused by an arbitrarily small spatial change in f.

Lastly, in defining continuity in a digital array, it may also be desirable to take account of sensor noise by recognising that small changes may be due to noise rather than to the signal f. Statistical modelling (typically gaussian) of the noise indicates what intensity differences are likely to be attributable to noise.

Definitions of continuity for digital arrays and methods of locating discontinuities have been suggested in the literature. Beattie (1981) essentially records any edge (a change in intensity between two adjacent pixels that exceeds some threshold) as being potentially the site of a discontinuity of f. Parallel edges of like sign (their intensity differences are in the same direction) are treated with suspicion, they may lie on an extended gradient of f, rather than on a discontinuity. Intensity gradients can be distinguished from discontinuities by using a local differentiation operator followed by "lateral inhibition" - subtracting off a local average response. Over an extended gradient the output

* "grey level" is the term used to refer to a single quantum of intensity, when intensity is represented as a binary word

of the differentiator is uniform, so that after lateral inhibition the result is zero. Marr's $\nabla^2 G$ operator (section 2.1) also inhibits output on gradients, in a similar way.

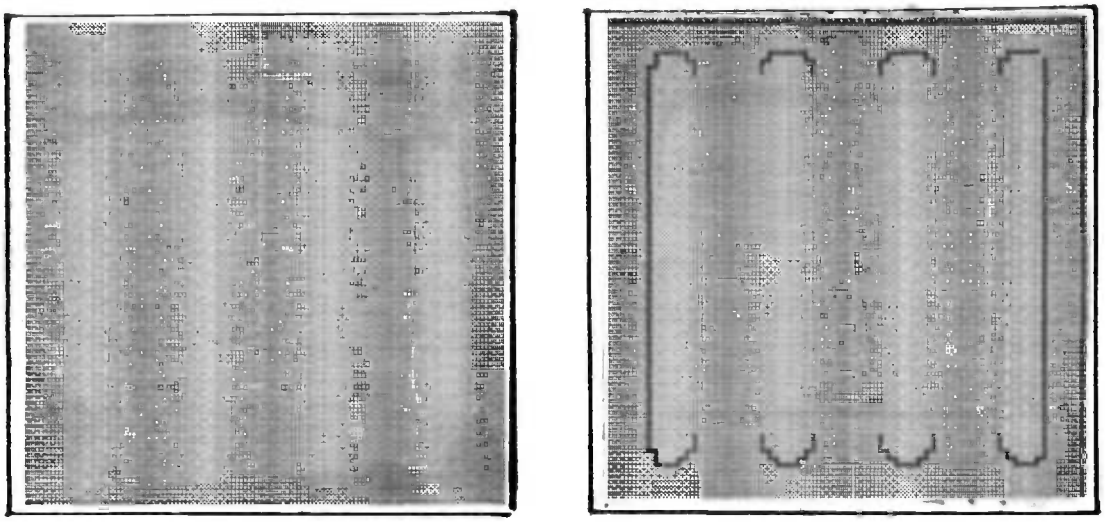
Binford (1981) argues that it is necessary to detect discontinuities both in f itself and in its gradient. At genuine discontinuities of f or its gradient, differentiation plus lateral inhibition produces a peak in output which marks the position of the discontinuity. Marking the zeros of the $\nabla^2 G$ operator does not however pick out isolated gradient discontinuities (fig 2.5). Haralick (1980) proposes a basically similar approach, using a local differentiation operator followed by peak detection, but in which the local operator is carefully based on least squares fitting of the pixels in a certain neighbourhood to a linear function (uniform gradient). The fitting is done by calculating, for each pixel, parameters α, β and γ that are estimates of the coefficients in

$$g(x,y)=\alpha x+\beta y+\gamma \qquad (2.2)$$

- the best fit linear function to f, in a small neighbourhood around the pixel. The next step is to calculate the statistical significance of the difference in (α,β,γ) across each pair of adjacent pixels. A local maximum of the significance measure is taken to indicate a discontinuity, which may either be a step or a gradient discontinuity or a combination of both. This approach incorporates the important idea of fitting the array data to a model, in this case a locally linear model. Hueckel's (1971) operator also performs least squares fitting, to a model that is restricted to step edges. Using either of these operators, only a limited number of discontinuities (one or two) can be detected in any one neighbourhood. If the data contains several edges within one neighbourhood either operator is likely to report an edge in the wrong position within the neighbourhood, or to report no edge at all The Hueckel operator also makes an assumption about local straightness of an edge. It is when the neighbourhood is large that these errors become significant Reference to global context in the array is necessary, however, to overcome the effect of noise of various kinds. The weak continuity relaxation scheme, proposed in this thesis, is a way of doing this without resorting to a large fixed neighbourhood.

2.2.2 Global context

Two main ways of referring to global context for locating discontinuities have been described in the literature. They are edge tracking (e g. Nevatia and Babu (1979), Ramer (1975)) and region analysis (e.g. Brice and Fennema (1970)).



a) triangular wave. b)

Fig 2.5 Zeros of the $\nabla^2 G$ operator do not mark gradient discontinuities as a & b show.

In edge tracking, edges are linked to form a continuous boundary. The boundary is extended from one end by successively searching for the most likely continuation, based on the initial output from a local edge operator. If the operator is an oriented detector, yielding angle information, then continuity of angle can also be taken into account in choosing a likely continuation. Tracking enables a long boundary to be detected in noise, even if its contrast is low.

Region analysis also starts with a local operator (a simple differentiator suffices) to define an initial set of boundaries throughout the image. The boundaries delineate a number of closed regions. Reference to global context is achieved by measuring various properties of the regions. A pair of regions may then be merged, dissolving its common boundary, if the properties of each region (area, perimeter etc.) and the relationship between them (e.g. contrast and length of the common boundary) suggest it. High contrast boundaries, and low contrast ones whose length is substantial compared with perimeters of the regions it divides, survive the merging procedure. As in edge tracking, a long boundary tends to be preserved, even if the contrast across it is low.

Neither of the methods - edge tracking and region analysis - is suitable for parallel array implementation. Edge tracking is essentially serial because it follows boundaries one edge element at a time; it cannot start on the next element until it has dealt with the current one There may be some scope for parallelism, however, in following several boundaries in an image at once, using several powerful processors but the task of scheduling the processors may not be easy. A cellular array processor with many simple cells, one per pixel, is certainly of no help there. Region analysis also admits some parallelism in that a merging operation can be carried out over several boundaries at once but again it is not clear how to implement such parallelism.

In order to implement the task of reference to global context in a parallel array manner the task can be specified in terms of the goal state of the array as a whole. This leads to a definition of continuity for digital arrays which simultaneously deals with the local notion of a discontinuity, discussed in section 2.2, and also with global context. The array data is fitted to a piecewise continuous model and the result is deemed to be continuous within the pieces and discontinuous on boundaries between pieces. A continuous fit is attempted *everywhere* in the array, but admitting failure in a few places (the discontinuities). This is similar in some ways to Haralick's local fitting scheme but here the fitting is done globally across the array, rather than locally within a neighbourhood.

In most of the work reported in this thesis, "continuous" is interpreted - rather strongly - as "constant". Fitting to constant valued regions is not an entirely general method because it fails to cope with steep extended gradients that can occur in real images. Generalisation of this interpretation of continuity is considered in section 2.4.

2.3 Edge-region duality

This section discusses a constraint which must be satisfied, under certain conditions, by region boundaries in an array. This constraint was used by the author in an early attempt to label discontinuities using relaxation. As we shall see this formulation is inadequate for the discontinuity labelling task but sheds some light on methods for the satisfaction of linear constraints by relaxation (chapter 5).

The constraint is essentially a mathematical relationship characterising the mapping between a set of regions of constant grey-level, represented as an array of numbers, and an array of strengths (differentiated intensities). The constraint does not contain any information, in itself, about a model of allowable scenes and images. Therefore it is only effective when combined with scene model constraints, in which case it allows them to be applied in "gradient space" rather than "image space". This will be useful where the scene model constraints are most naturally expressed in gradient space

The motivation to develop and investigate the edge-region duality constraint came from the Hanson and Riseman (1978) edge relaxation algorithm. The spirit of their algorithm is to impose some sort of continuity on an initial distribution of edge strengths. It is based on the idea of "good line continuation" - roughly that good boundaries (lines) are unbroken ones. They extend this idea to cover edges labelled with strengths. An edge's six neighbours (fig 2.7a) are inspected its strength is enhanced if it forms part of a continuous chain of strong edges, and diminished if it is isolated. Fig 2.6 shows an example of the effect of this, taken from an implementation, in the PARAPIC language, of a similar algorithm (appendix E), for parallel array processor. Their algorithm has drawbacks however:

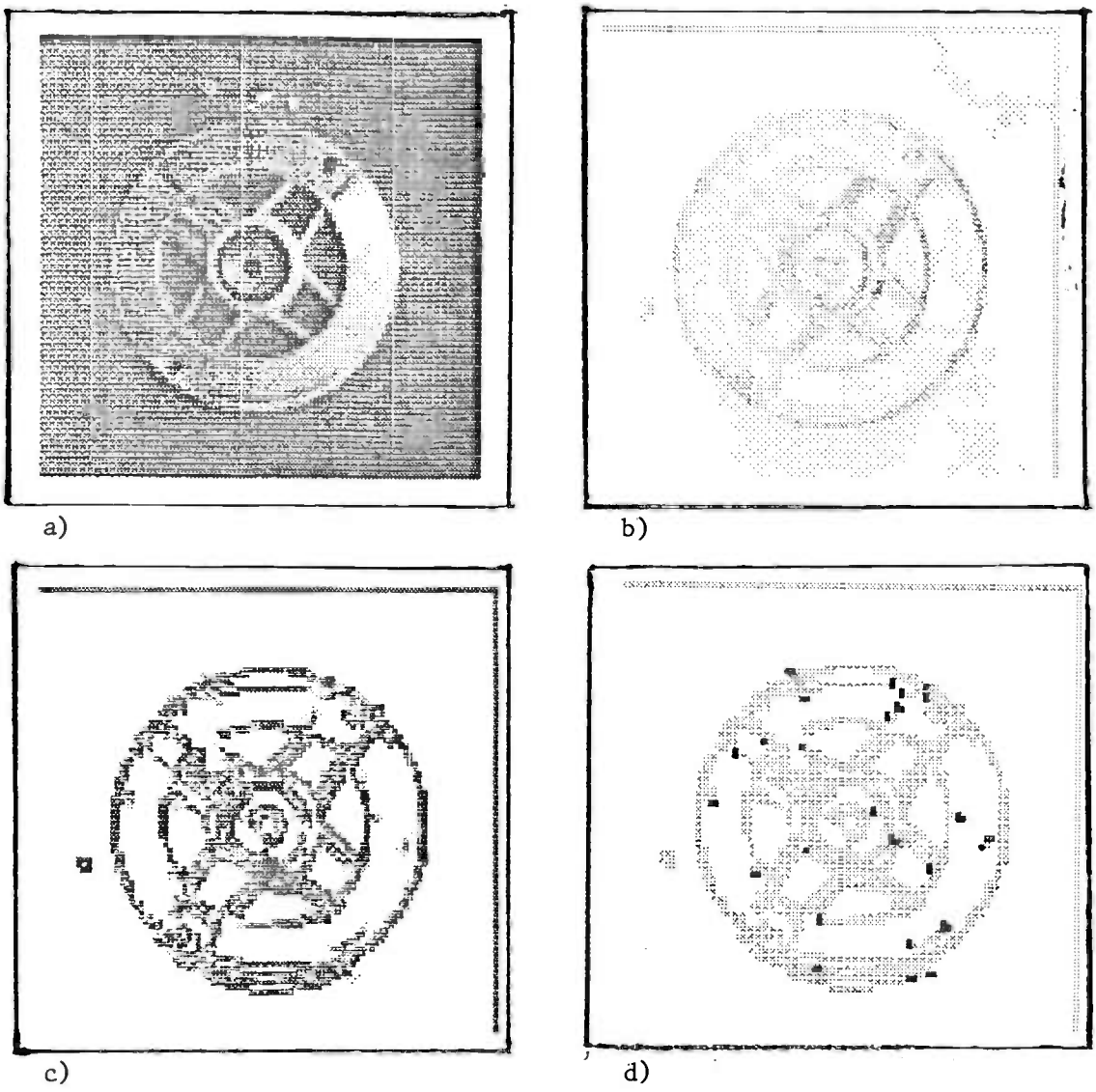
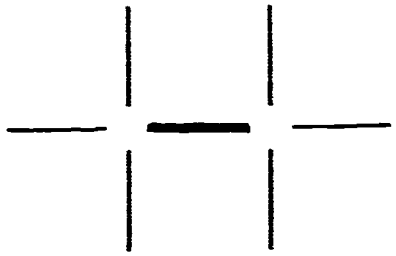
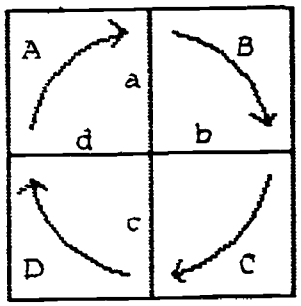


Fig 2.6 A close relative of the algorithm of Hanson and Riseman, implemented in PARAPIC, is applied to image a). An initial estimate of edge strengths is made with a differentiation operator (b) and is enhanced by the relaxation algorithm to produce (c). The convergence of the algorithm is not guaranteed - in this case it ends up oscillating. Dark points in (d) mark those pixels where oscillation occurs.



a)



b)

Fig 2.7 Edge-region duality. a) An edge - the line dividing two adjacent pixels - has 6 neighbours. b) Following any closed path in an array, the total change in grey-level is obviously zero. In the particular case of a closed path around four adjacent pixels, this leads to a local constraint on edge strengths. (The strength of an edge is simply the difference in grey-level of the pixels on either side.) The strengths on edges a,b,c,d sum to zero.

1. Constraints on edges are not explicitly stated. The spirit of the algorithm is to perform some sort of edge cleaning operation, in an iterative fashion, but there is no clear definition of the problem that is being solved.
2. At each iteration, new edge strengths are computed from current edge strengths, by a non-linear formula, following Rosenfeld et al. (1976). Analysis of the effect of the repeated application of a non-linear formula is difficult, and was not attempted by Hanson and Riseman. They show two examples in which edge strengths appear to converge, but there is no guarantee of convergence, and oscillation has been observed in the PARAPIC implementation (Blake 1981) (fig 2.6d).

2.3.1 The duality constraint

An image can be partitioned into a number of regions - sets of connected pixels - in which case the boundaries between adjacent regions are chains of edges. Conversely, edges extracted from an image may form boundaries which delineate a number of regions. This duality is used here to derive constraints for edges. We will regard a set of edges as consistent if it corresponds to a set of regions, in which each region is of uniform grey level, so that the common boundary between two regions is of uniform strength. (Each edge is labelled with a strength which represents the difference in grey level of the regions on either side.) This requires edges to be continuous; they may not have gaps, or other discontinuities in strength. Representing a scene by a set of regions with uniform grey level is something of an over-simplification, but may be reasonable under diffuse lighting, with lambertian surfaces, and when reflectivity is roughly constant within each region. Horn (1974) calls this simplified model "Mondrian World". He derived an algorithm (lightness computation) for eliminating the effect of illumination gradients in scenes under this model.

An initial set of edges, obtained from an image by differentiation, already satisfies the continuity constraints, as we will see. However, when additional constraints are imposed then it is necessary to ensure that the edges also continue to satisfy the continuity constraint. For instance, an image often contains many spurious, weak edges due to noise from the camera and from quantisation, and as a result of small variations in surface reflectance. We wish to retain only those edges caused by physically significant events such as object boundaries, which are likely to appear as substantial changes in image intensity. This can be achieved by a "minimum strength constraint": no edges may exist where the

magnitude of the grey level difference between adjacent pixels in the *original image* falls below a certain threshold. The threshold itself should ideally be determined from sensor parameters. If this is not practicable it could be derived from a histogram of initial edge strengths, set to suppress a certain fixed proportion of the edges.

There may be other ways of picking out important edges than applying the minimum strength constraint. One alternative that has been tried by the author involves applying a non-linear transformation to the initial edge strengths, to enhance the strong edges and diminish the weak ones. In general the result no longer satisfies the continuity constraints but may be adjusted to do so by relaxation. However, the minimum strength constraint is stronger, because it *eliminates* edges from substantial areas of the image - the relaxation process takes place only in the remaining areas. The result is a sparse distribution of edges and consequently few but large regions, free of edges with unattached ends.

2.3.2 Local constraints

The edge-region duality constraint has been described qualitatively. Now we show how this leads to a set of local constraints on the array of edge strengths First some terms must be defined more precisely.

An *image* is an array of positive numbers.

$$I_{i,j}\ ,\ i=1..N,\ j=1..N, \tag{2.3}$$

in which each element represents the intensity of light falling on one picture element (pixel) and is called the "grey-level" of that pixel. An *edge* is the dividing line between two adjacent pixels A *boundary* is a chain of edges, connected end to end The edge strength array has two components, the horizontal and vertical *edge strengths:*

$$H_{ij} \text{ and } V_{ij}\ ,\ i,j=1..N. \tag{2.4}$$

Strengths are signed, to distinguish a light-to-dark transition from a dark-to-light transition, and their magnitude indicates the absolute difference of intensity across the edge. Finally, edge strengths (H,V) are said to *represent* an image I when $\forall i,j=0,..,N+1$

$$I_{i,j}-I_{i-1,j} = H_{i,j} \text{ and } I_{i,j}-I_{i,j+1} = V_{i,j} \tag{2.5}$$

- this means that (H,V) are simply differences of intensity between adjacent pixels of I where, on the array boundary (i=0 or i=N+1 or j=0 or j=N+1), I_{ij} is taken

to be zero.

The duality between regions and edges - that a set of edge strengths (H,V) must correspond to a set of regions of uniform intensity - requires that, at the very least, (H,V) must represent some image I, because a set of regions, each with a uniform grey level, is itself an image. The strengths (H,V) have $2N^2$ degrees of freedom but an image, I, has only N^2 degrees of freedom. We expect, therefore, that edge-region duality imposes N^2 constraints on (H,V). A set of N^2 constraints can be derived directly from the definition of representation (2.5):

$$\forall i,j \quad S_{ij}=0 \text{ where } S_{ij} = H_{ij}-V_{ij}-H_{i,j+1}+V_{i-1,j} \tag{2.6}$$

Each edge is linked, by one or other of these constraints, to its six immediate neighbours (fig 2.7a). The meaning of the constraints is that, over each of N^2 small closed paths (fig 2.7b) the strengths of edges crossed sum to zero. In fact this is true for any closed path This result is directly analogous to Stokes' theorem in vector calculus, as the following table shows:

Discrete array	**Vector field**
Image I	Scalar field φ
Edge strengths (H,V)	Vector field $A=\nabla\varphi$
Constraint on (H,V)	curl(A)=curl($\nabla\varphi$)=0
implies that	implies that
strengths sum to 0 over a closed path	$\int A.dl=0$ over a closed path (Stokes' theorem)

2.3.3 The principle of least disturbance

Edge-region duality, with the minimum strength constraint, does not yet completely specify edge strengths (H,V) for a given image I, but places a partial restriction on them. As we have seen, edge-region duality reduces the dimensionality of the space of (H,V) from $2N^2$ to N^2. The minimum strength constraint reduces this further by fixing some of the edges at zero strength. Within this cut down space, it matters greatly just which solution is chosen. For instance, for any image I, an edge strength map (H,V) s.t.

$$\forall ij \quad H_{ij}=0 \text{ and } V_{ij}=0 \tag{2.7}$$

is always consistent (lies within the cut down space) but is clearly inappropriate as a set of edge strengths. Therefore the "least disturbance" principle is invoked, choosing that set of consistent edge strengths (H,V) that is closest (in

euclidean distance) to the initial estimate $(H^{(0)}, V^{(0)})$, defined by:

$$H_{ij}(0)=I_{ij}-I_{i-1,j} \text{ and } V_{ij}^{(0)}=I_{ij}-I_{i,j+1}. \tag{2.8}$$

This generates an optimisation problem to minimise

$$T= \|H-H^{(0)}\|^2+ \|V-V^{(0)}\|^2 \tag{2.9}$$

subject to the constraints on (H,V), and where

$$\|H-H^{(0)}\|^2 = \sum_{ij} (H_{ij}-H_{ij}^{(0)})^2,$$

the euclidean norm. Chapter 5 shows how the solution of this problem can be achieved by relaxation in a parallel, iterative way.

The effect of the relaxation process is to coax the initial estimate of the labels (edge strengths) into satisfying the constraints, by the most direct available route. The labelling achieved is that *consistent* set of labels which is closest to the initial set. The initial labelling is disturbed only insofar as is necessary to achieve consistency. In this way, information in the initial labelling is preserved as far as possible It can be shown that this process is equivalent to a linear projection from the $2N^2$ dimensioned space of edge strengths to the lower dimensioned space of consistent edge strengths (appendix C.2).

The use of a euclidean measure of distance needs some justification - other measures have been suggested in the literature (Peleg and Rosenfeld (1981)). Minimisation of euclidean distance is appropriate for use with data which approximates to some model, and in which the deviation from that model is normally distributed. In that case the minimal labelling is a "maximum likelihood estimate" (Morrison (1976)). In the context of edge strengths, the relevant model is the flat region model described earlier. Deviation from the model (assuming scene conditions are such that the model is appropriate) will arise because of sensor noise, which is likely to be normally distributed, and also from surface texture, of unknown distribution. In the case of texture, if the scale of the texture is small enough, the central limit theorem says that the normal distribution can be used (Hays and Winkler (1970)). The scale of the texture should be small compared with the areas over which the model fitting process takes place. For edge strengths, a single section of boundary between two regions must, as we saw earlier, have uniform strength. Therefore the determination of the edge strength of that section is effectively done by a single fitting process. If the boundary is long enough that the texture is not highly correlated over the length of the boundary then the normal distribution, and hence the use of least

squares fitting, is appropriate.

If the texture is an uncorrelated, stationary process the texture signals on each of the pixels are independent, identically distributed random variables. In this case the sum, over a neighbourhood, of the pixel signals is approximately normally distributed, if there are enough pixels in the neighbourhood. This result still stands when there is some degree of correlation between pixels, provided the correlation distance - the distance above which inter-pixel correlation is negligible (as measured by the co-occurrence matrix (Haralick et al. (1973)) - is small compared with the diameter of the region. There is another point that requires some comment: the euclidean distance measure in (2.9) is in edge space (gradient space). Therefore the sensor noise should ideally be independent and normally distributed in edge space, not in image space. However the sensor (e.g. a vidicon camera) produces noise which can probably be modelled as being independent, normally distributed in image space. In edge space therefore there will be some degree of correlation between the noise on adjacent edges. Just how much correlation depends on the exact configuration of the edges themselves. The least squares fitting occurs in the restricted area allowed by the minimum strength constraint; where two edges in that area are neighbour to a common pixel the noise signals on those two edge will be correlated.

2.3.4 Discussion

Two constraints - edge-region duality and minimum strength - have been described. They can be applied by relaxation (see chapter 5) and the resulting algorithm is related to Horn's algorithm (Horn 1974) for determining lightness (perceived brightness) in an image. His algorithm fits intensity data to a piecewise constant model, just as with the algorithm described here. The difference between the two algorithms is that Horn uses the constraint on the intensity I that within each constant region:

$$\nabla^2 I = 0 \textit{ within a region} \tag{2.10}$$

whereas here the corresponding constraint (minimum strength) is that (H, V) is zero within regions, that is:

$$\nabla I = 0 \textit{ within a region} , \tag{2.11}$$

In fact this equation (2.11) accurately represents constant intensity within regions, whereas (2.10) is a weaker condition, necessary but not sufficient. The constraint (2.11) is applied in edge space, so the edge-region duality constraint

must simultaneously be satisfied, to ensure that (H,V) remains consistent.

The results of imposing these two constraints are shown in fig 2.8. Fig 2.8c shows the final labelling (H,V) and from this a reconstruction of the underlying image (fig 2.8d) is produced by integration of either H or V - in fact inverting (2.8). The reconstructed image corresponds to Horn's perceived lightness image.

Another advantage of this algorithm compared with Horn's arises because it operates in edge space, and non-zero edges are sparse. Relaxation need not be applied in those areas which are constrained to zero strength - the interiors of regions, which comprise the majority of the image area. Therefore the algorithm may be executed more efficiently, although this increase in efficiency is not realised on a parallel array processor in which all operations must be performed throughout the array.

However, in chapters 2.1 and 2.2 the purpose of edge computation in early vision was said to be to establish the positions of discontinuities. The enforcement of the two constraints described here cannot be regarded as an adequate procedure for doing that. This is because the positions of the discontinuities are largely established by the minimum strength constraint, prior to relaxation. The minimum strength constraint acts as a boundary condition to relaxation, rather than being propagated during relaxation. It therefore chooses the positions of the discontinuities on a purely local basis. Global reference was made only subsequently, by relaxation, to impose the edge-region duality constraint; the effect here is simply to remove boundaries with unattached ends (because they are not region boundaries) and to do some redistribution of strengths amongst the remaining edges. In fig 2.9 the effect of imposing minimum strength and edge-region duality is shown on a less clearly defined image than in fig 2.8. There is an important missing boundary. It was removed by the minimum strength constraint because the contrast across it was low. It should have been retained however because it is long and divides two major regions. Global reference could reveal that. The next section explains how this can be achieved by using a weak continuity constraint.

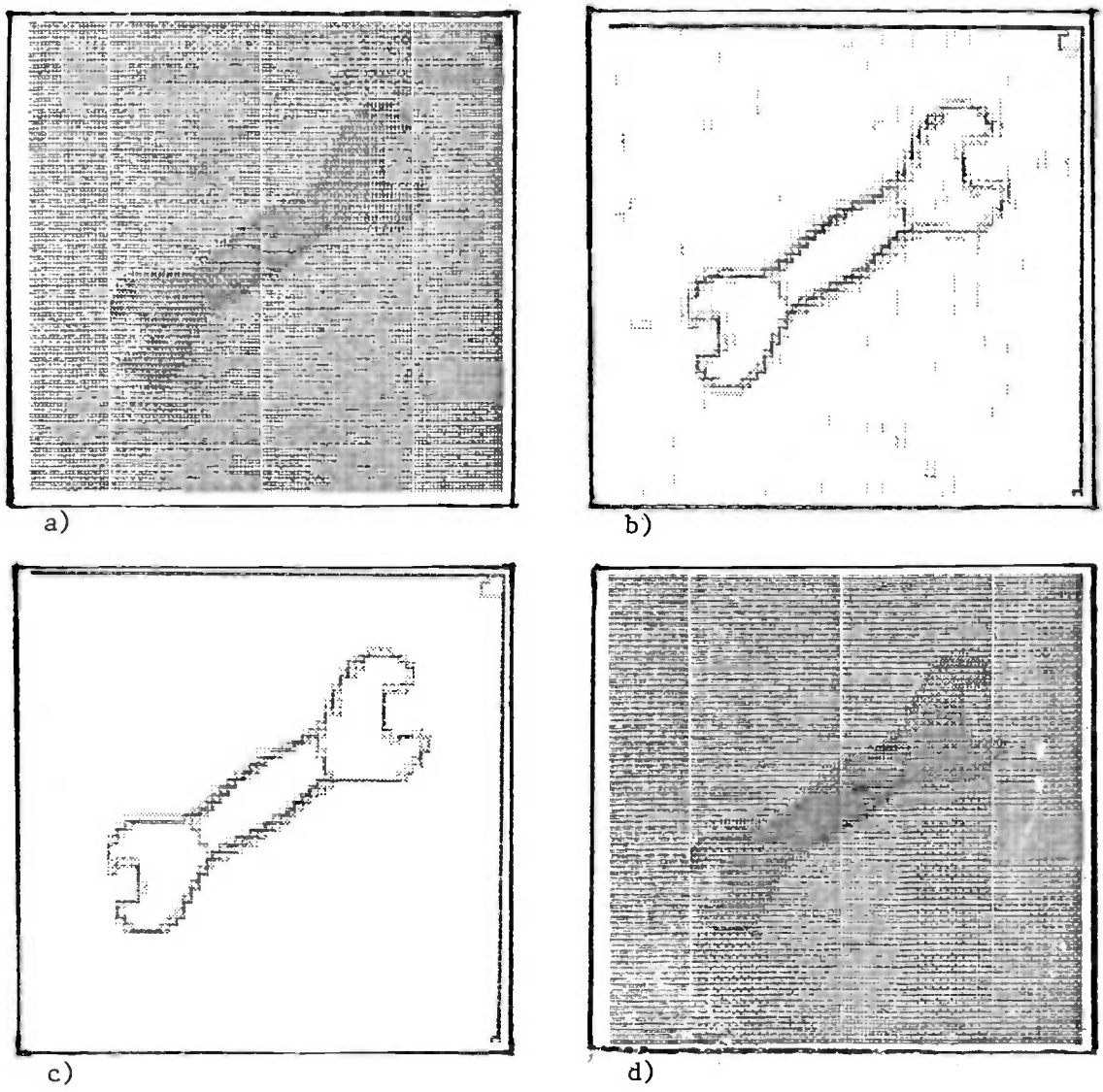


Fig 2.8 Edge-region duality and minimum strength constraints applied to (a): initial edge strengths (b) are obtained by differentiation and subjected to relaxation (c). An underlying image (d) can be reconstructed by an integration process, from (c).

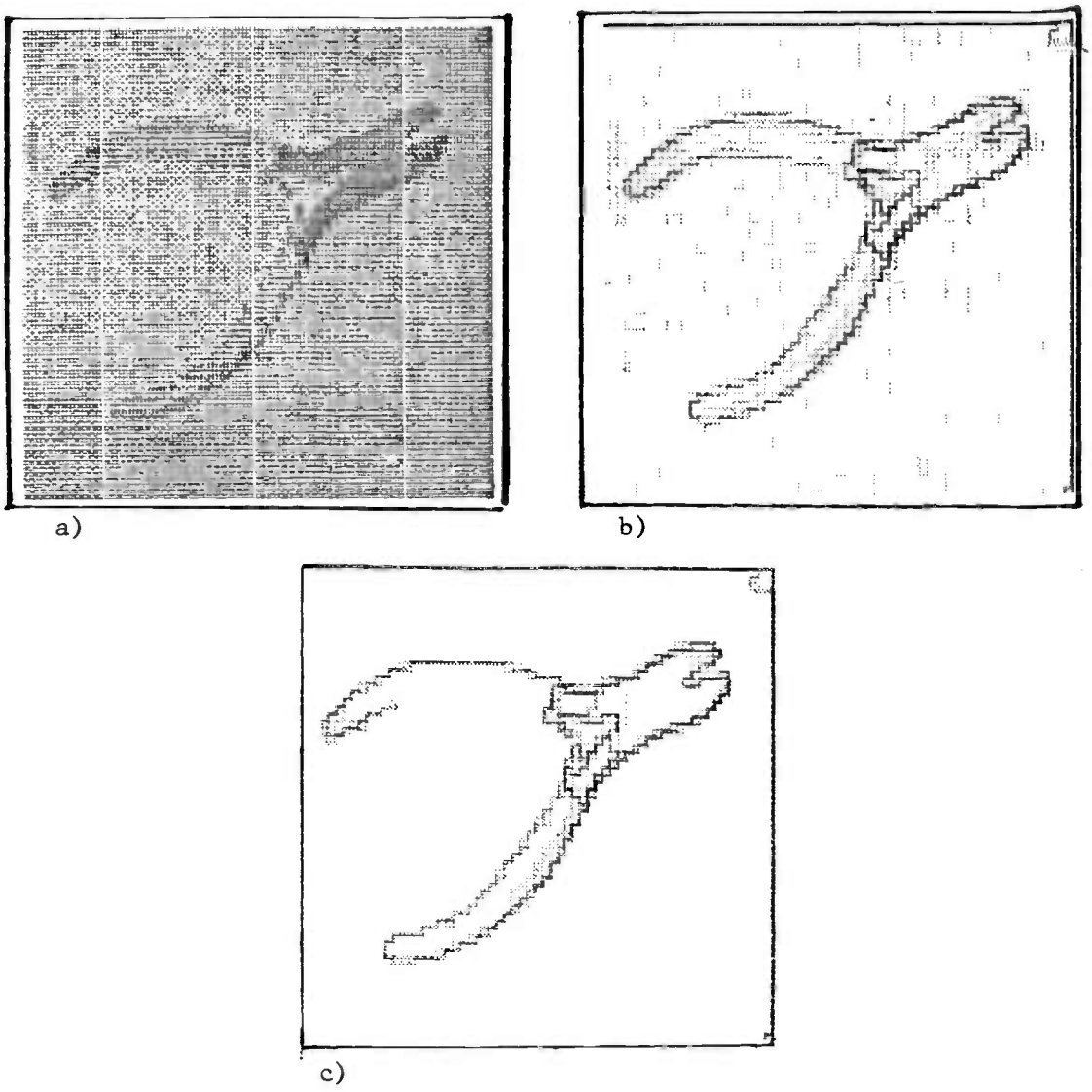


Fig 2.9 As fig 2.8 but with a different image. It is apparent in (c) that a major line along the handle has been missed.

2.4 Weak continuity

In chapter 2.1 it was proposed that each image in a stack of intrinsic images should be subject to the constraint that it be continuous almost everywhere. We formalise continuity "almost everywhere" as a weak continuity constraint. A weak constraint is one that may be broken at a cost - a penalty must be paid for its violation. This thesis considers, in particular, the application of a weak constancy constraint to the intensity image, and to the angle image (defined in section 2.1.3) for labelling lines and curves. Weak constancy - that the array of values is constant almost everywhere - is a rather strong interpretation of continuity, and too strong to be always applicable, but still surprisingly useful on real images. A more general definition of continuity is considered later in this section.

As with edge-region duality and minimum strength constraints (chapter 2.3), weak continuity constraints do not totally determine the labelling. The final choice of labelling is made according to the least disturbance principle, described earlier. This can be illustrated in a simplified labelling problem: to subject a one dimensional array of values to weak constancy constraints. Adjacent elements of the array are constrained to have the same value - or a penalty is paid. Discontinuities in the array are then deemed to be those places where the constancy constraint is broken.

2.4.1 Weak constraints and non-convexity

Consider an array of N elements, each labelled with a value $x_i, i=1..N$, initially $x_i^{(0)}$. Each pair of elements must either be made equal ($x_i=x_{i+1}$), or else the junction of the two elements is labelled as "discontinuous" and a fixed penalty, c^2, is incurred. The least disturbance labelling under these constraints, is obtained by minimising an objective function:

$$F = \sum_i (x_i - x_i^{(0)})^2 + c^2 \times (number\ of\ broken\ constraints) \qquad (2.12)$$

- a distance measure on $\mathbf{x}-\mathbf{x}^{(0)}$ plus a penalty of c^2 for each broken constancy constraint.

The penalty constant c gives control over the reluctance to allow discontinuities. A high value of c results in a broad brush segmentation marking only the principal discontinuities, whereas a smaller value results in the inclusion of more detail.

Unfortunately, the optimisation problem that the weak constraint generates has the mathematical property of non-convexity. This means that, unlike problems in which the constraints are linear (e.g. section 2.3) and which therefore have a unique local optimum, there may be many of them, so that a continuous search for a local optimum does not guarantee locating the required global optimum. To illustrate, consider again adjusting values in an array. Fig 2.10 shows three different arrangements of an array. Starting with fig 2.10a, continuously and simultaneously adjusting the values to reach fig 2.10c, the array passes through intermediate states like fig 2.10b. In fig 2 10b, the constancy constraint is broken once, incurring a penalty, c^2, while in fig 2 10c the values are all equal - no penalty. Let us assume that in fig 2.10c, the remission of the penalty outweighs the increase in the distance measure in moving from fig 2.10a to fig 2.10c, so that fig 2.10c is a more desirable state than fig 2.10a. Then clearly, fig 2.10b bears both a penalty and an increased distance measure, and is therefore less desirable than either of fig 2.10a or fig 2.10c. Fig 2.10c cannot be reached by following a strategy of continuous improvement and this is precisely what is meant by "non-convexity".

It might be tempting to think that the non-convexity is due to an unfortunate expression of the problem, and that some transformation could be found to make it convex as, for example, in fig 2.11. This is not the case. The non-convexity is a fundamental property of the problem. To see this, consider a general transformation of the problem, a co-ordinate transformation

$$x' = T(x). \tag{2.13}$$

The transformation T should be continuous and invertible - a diffeomorphism (Poston and Stuart (1978)) - to ensure a non-degenerate representation in the new coordinate system. However the non-convexity of $F(x)$ occurred because of discontinuities in F (as defined in (2.12)). and any discontinuous function on an array is non-convex (Roberts and Varberg (1976) p 93.). In the new coordinate system $F(x)$ becomes $F'(x')$ where

$$F = F'.T. \tag{2.14}$$

But T is continuous so F' must be discontinuous. (The proof is by reductio ad absurdum: if F' were continuous everywhere and since T is continuous, then from (2.14) F would be continuous everywhere - in contradiction to the initial hypothesis.) Therefore the discontinuity and hence the non-convexity persist in the new coordinate system.

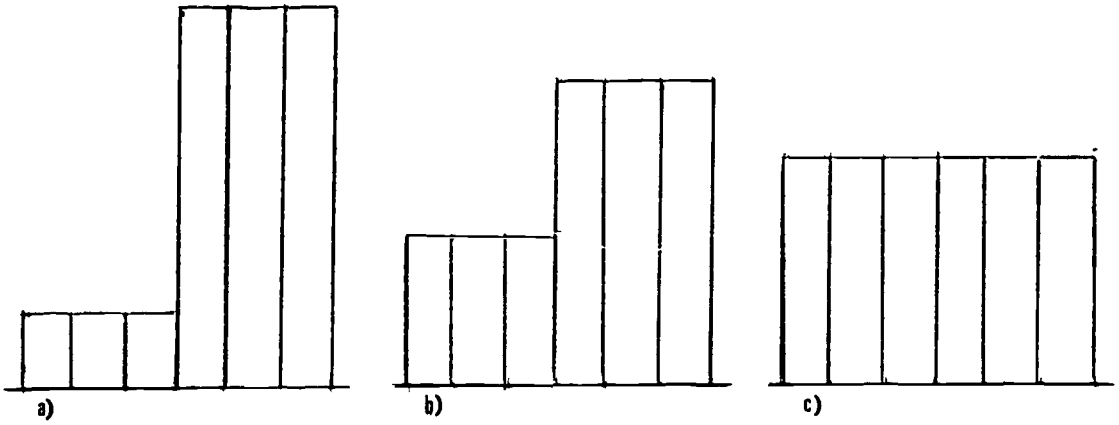


Fig 2.10 A demonstration of non-convexity in weak constancy relaxation, in a six element linear array. Assume that the configuration of (c) incurs a lower cost than the initial configuration (a): it is not possible to follow a path of continuously decreasing cost from (a) to (c) - (b) has greater cost than either. See text for details.

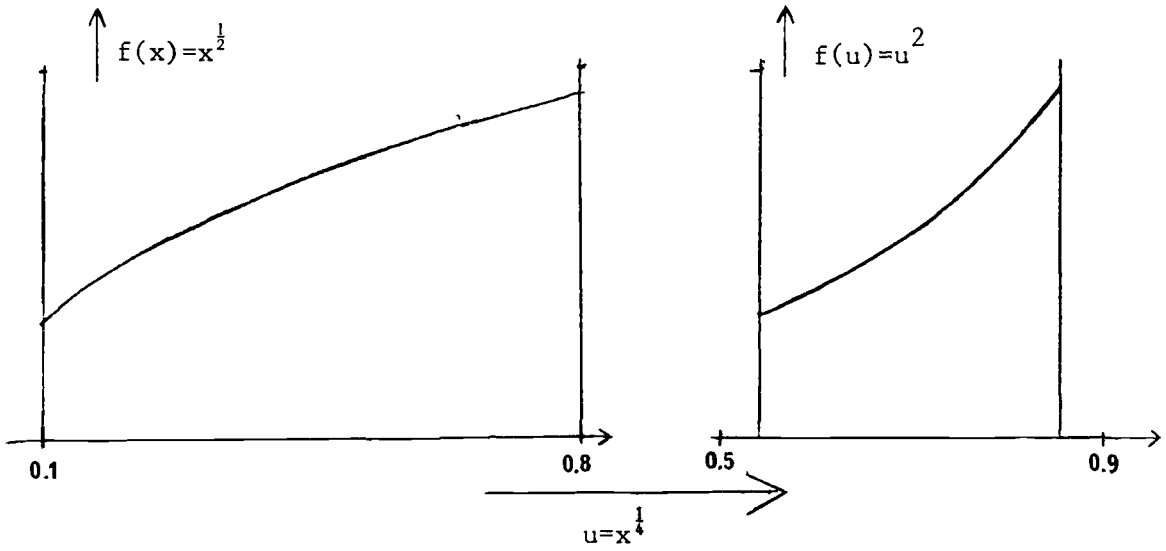


Fig 2.11 An example of a coordinate transformation that converts a particular non-convex function into a convex one.

The consequence of non-convexity is that a strategy of continuous improvement is inadequate, but what are the alternatives? Berthod (1982) and Ullman (1978) both suggest settling for a local optimum as the best available solution to a non-convex problem. This is quite inapplicable here, however, because the initial labelling is already a local optimum of F in (2.12). Hill-climbing would make absolutely no change to the initial state. Another possibility is to use a search, in which numerous configurations of the array are tried, but the number of configurations to examine is related exponentially to the number of elements in the array The search is impractical unless it can be guided in some way.

Instead the objective function F can be replaced by some other function F^* (constructed as a sum of certain convex envelopes) which is convex and which touches F in some places. A sequence of functions, beginning with F^* and ending with F is used to reach an optimal solution, where possible, or else to find a good sub-optimal one. This is called the "graduated non-convexity method" and is described fully in chapter 4.

2.4.2 Weak constancy

We now show and comment on the effect of the weak constancy constraint applied to various examples of data. The simplest example is a single step in a one-dimensional array. This is shown (represented as a histogram) in fig 2.12. As the penalty c^2 increases, a point is reached at which minimum cost is achieved by levelling the histogram. Beyond this point the cost of breaking the constancy constraint becomes greater than the cost of altering the data Fig 2.12 shows the theoretical behaviour (the optimisation has been solved correctly). The result using the graduated non-convexity method in this case finds the same optimal solution (although in general the method may produce a suboptimal solution). It is shown in appendix D.2 that in fact the single step is a special case for which it can be proved that a qualitatively correct result will be obtained by this method (correct except that the value of c is treated as if it were larger than it really is, by a factor of $\sqrt{2}$).

Fig 2.13 is another simple example, using graduated non-convexity, but this time two dimensional. As the penalty c^2 increases, first smaller dimmer rectangles and then larger brighter ones are levelled with the background. The criterion for levelling (for an isolated rectangle in an infinite array) is easily shown from the definition of the optimisation problem in (2.12) to be that:

$$area \times (contrast)^2 < c^2 \times perimeter, \qquad (2.15)$$

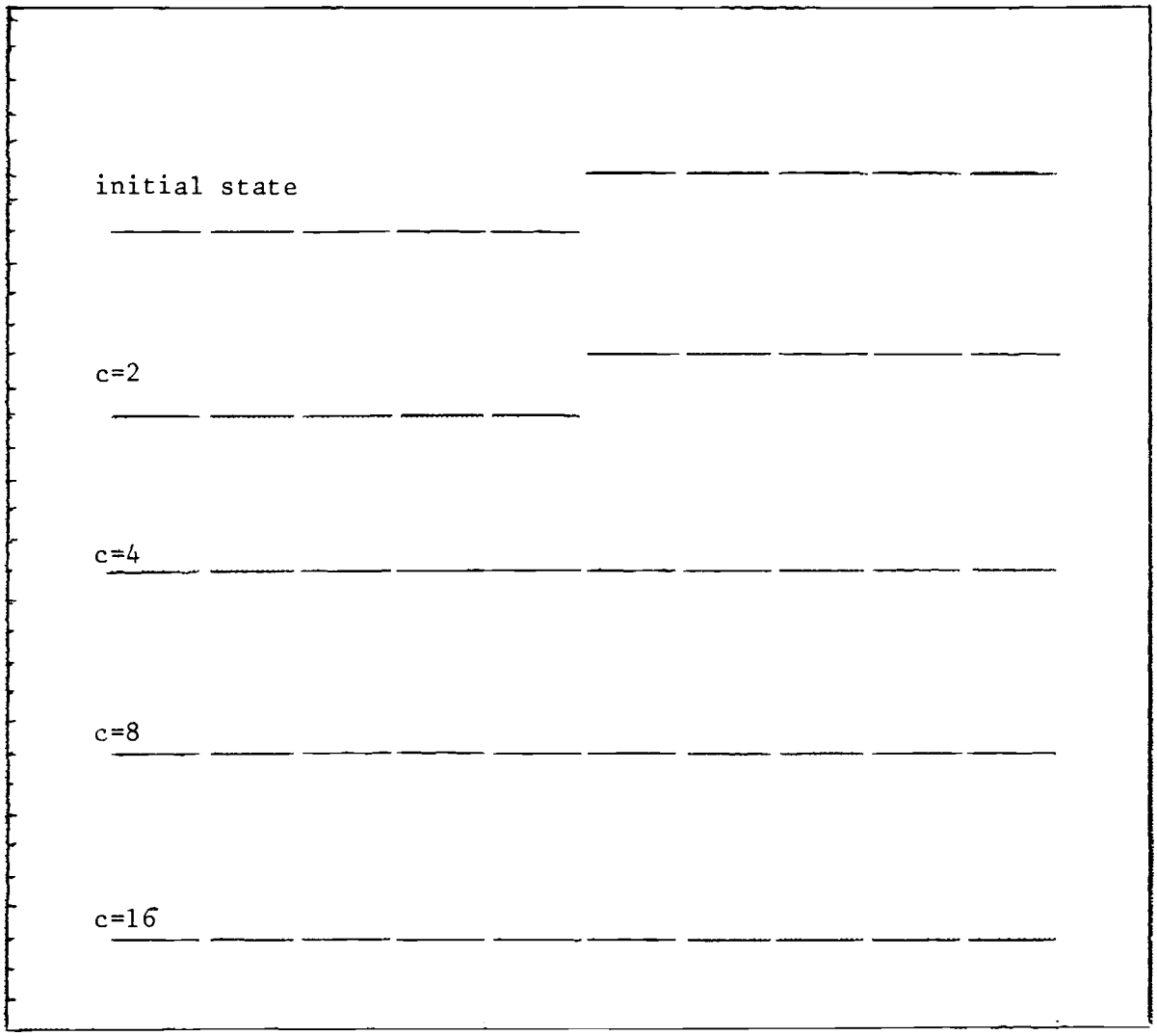


Fig 2.12 Weak continuity relaxation applied to a ten-element, linear array: for a penalty value, c, below a certain value a step discontinuity (of a given, fixed height) persists after relaxation. Above that value of c, the step is levelled.

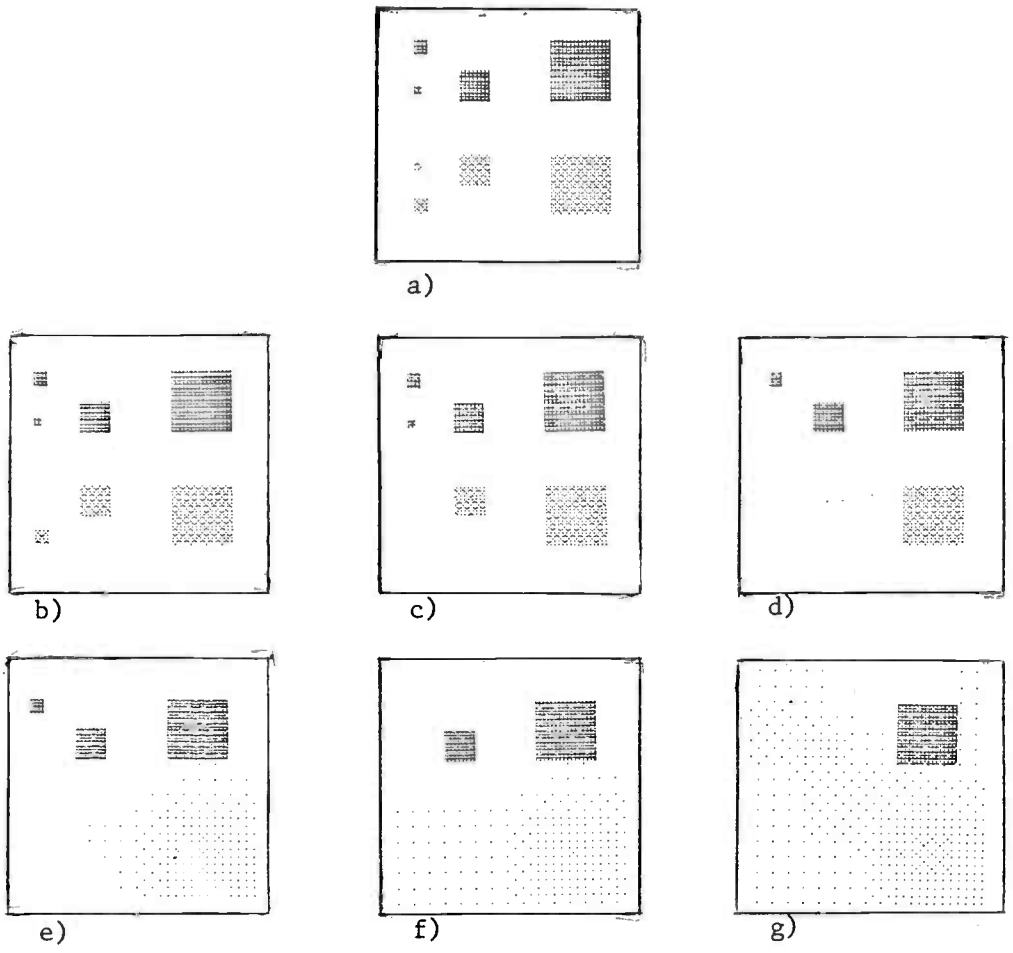


Fig 2.13 A test-card of squares, of different sizes and contrasts (a) and on a uniform background is subjected to weak constancy relaxation. As the penalty, c, for breaking a constancy constraint increases more squares are flattened ((b)-(g)) → first to go are the small/low-contrast ones and then as c gets larger, those that are large/high-contrast.

where the length of the perimeter is in fact the number of constraints broken if the rectangle is not levelled. The threshold (value of c) for levelling is thus a combined function of contrast, area and perimeter:

$$c_{thresh} = \left[\frac{area}{perimeter}\right]^{\frac{1}{2}} \times contrast. \qquad (2.16)$$

Lastly, two examples using grey level images: Fig 1.3 (in chapter 1) showed the result of imposing weak constancy on an intensity image - effectively fitting it to a piecewise constant model. More examples, with different images, appear in appendix F.

Finally, fig 1.4 (in chapter 1) showed weak continuity applied to an angle image, to arrange sequences of strokes into straight lines. This is done in the same way as with the intensity image, except that continuity constraints are imposed on angles only along lines of intensity discontinuity. Elsewhere the angle image has no meaning anyway. Again, there are further examples in appendix F.

2.4.3 Some variations and extensions of weak constancy relaxation.

We have, so far, used constancy constraints to represent continuity but this is too strong in general Weak constancy relaxation performs correctly on various configurations of step edges, including 2 4c that was treated erroneously by the $\nabla^2 G$ operator. On steep, extended gradients however, spurious lines of discontinuity are produced, running perpendicular to the line of greatest slope and this is shown on artificial images and on a real image, in fig 2.14.

One possible solution to this problem might be to use a more general model for regions - for instance a planar or a harmonic model. (A harmonic function f is one for which $\nabla^2 f = 0$. The set of harmonic functions includes all planes) An example of an image subjected to weak harmonic constraints is shown in fig 2.15, in which the gradient discontinuities in the processed image are marked. However, this is less satisfactory than weak constancy for edges that approximate to step edges. Instead of fitting the best step edge to the data, the edge appears as a short steep gradient and consequently the precise position of the edge is uncertain within the limits of the width of the gradient. Moreover, neither the planar nor the harmonic model is adequately general because curved surfaces in real scenes may give rise to higher order intensity distributions. Using correspondingly higher order constraints (the harmonic and planar constraints are second order) is not a good solution as the precision of positioning

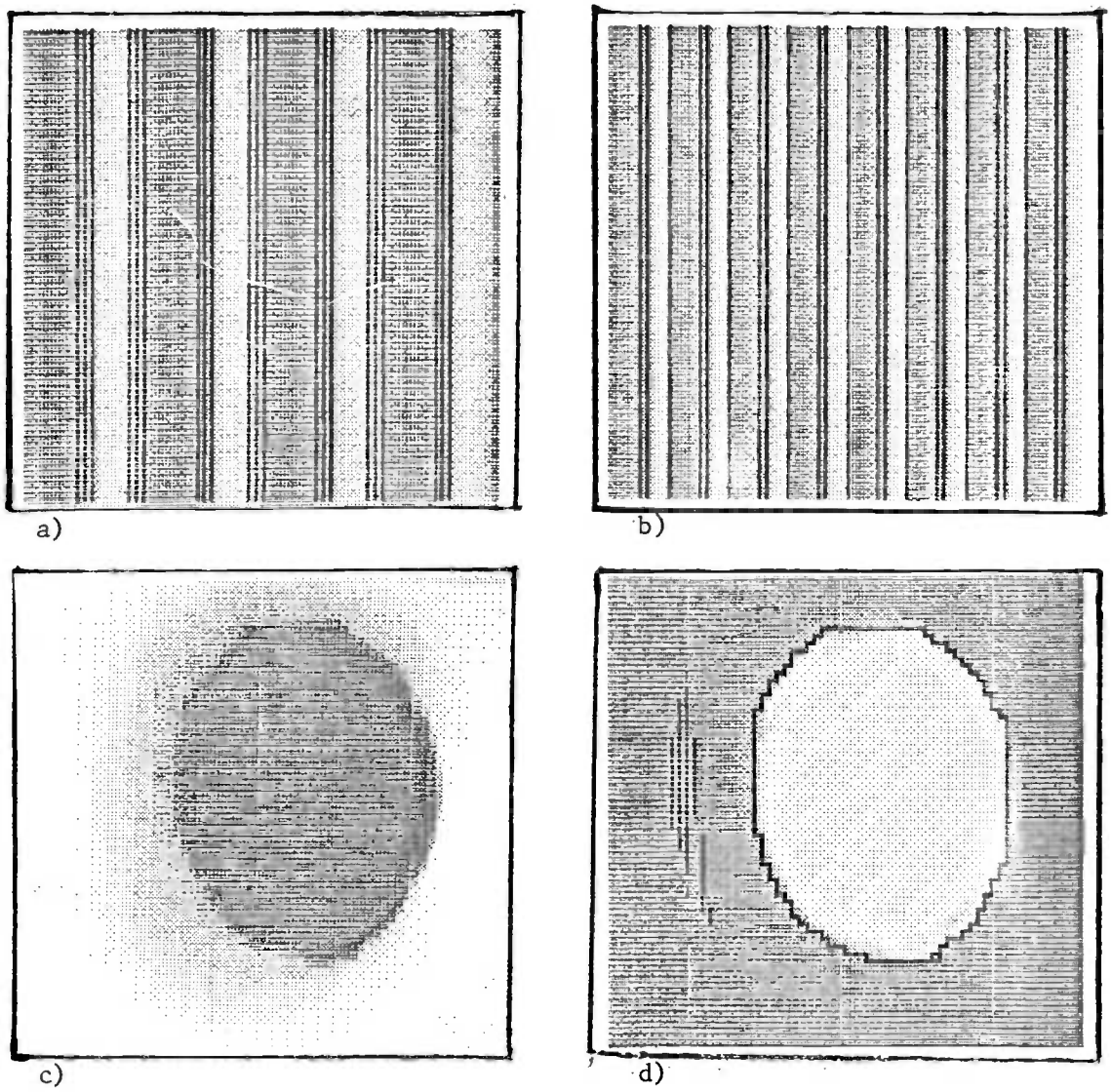


Fig 2.14 Weak constancy relaxation followed by marking of all remaining intensity changes labels step edges correctly. However the test figures in 2.3a and 2.4a, which have steep, extended gradients, produce some spurious lines ((a) & (b)). A real image (c) with an extended gradient exhibits the same effect (d).

of step edges becomes even worse.

A better approach might be to fit minimum energy surfaces (as for instance Ikeuchi and Horn (1980) and Terzopoulos (1982)) but as a weak constraint: any pair of adjacent array elements are either attached by a spring or else a fixed penalty is paid. For a one dimensional data array x_i this is expressed as a minimisation of the cost function:

$$F = \sum_i a\,(x_i - x_i^{(0)})^2 + \sum_i h_i (x_i - x_{i+1})^2 - c^2 \sum_i h_i, \qquad (2.17)$$

where h_i takes the value 0 or 1 and when 0 marks the position of a discontinuity. As before c^2 is the penalty for allowing a discontinuity and within the boundaries defined by those discontinuities, minimum energy functions are constructed (the second term of (2.17) does this). The generalisation to two dimensions is straightforward. The constant, a, controls the strength of the restoring force back towards the initial data. Horn and Schunk (1980), in their surface fitting algorithm suggested that the equivalent constant in their scheme be determined by the expected noise content of the data. The minimisation is done over $\mathbf{x}$ and $\mathbf{h}$ - a mixed real and integer programming problem. The integer valued $\mathbf{h}$ automatically makes the problem non-convex, and it remains for future effort to try to adapt the graduated non-convexity method of chapter 4 to cope with this.

Just as weak constancy relaxation was applied to a local angle data array, to find straight lines and the vertices between them, so this scheme for constructing minimum energy functions is also applicable to angle data. The result is a parallel algorithm for constructing minimum energy curves in an image and labelling the vertices/discontinuities between them.

Barrow and Tenenbaum (1981) showed that construction of minimum energy curves could also be useful for reconstructing a 3-D curve from its 2-D projection. Weak, minimum energy, angle relaxation in 3 dimensions (and consequently with two local angle at each point) could be used to do that, in parallel, and would be able to label vertices between 3-D curves also.

Finally we consider the effect of modifying the form of the distance measure used in the cost function. Instead of the distance term

$$\|\mathbf{x} - \mathbf{x}^{(0)}\|^2$$

a similar measure in edge space could be used:

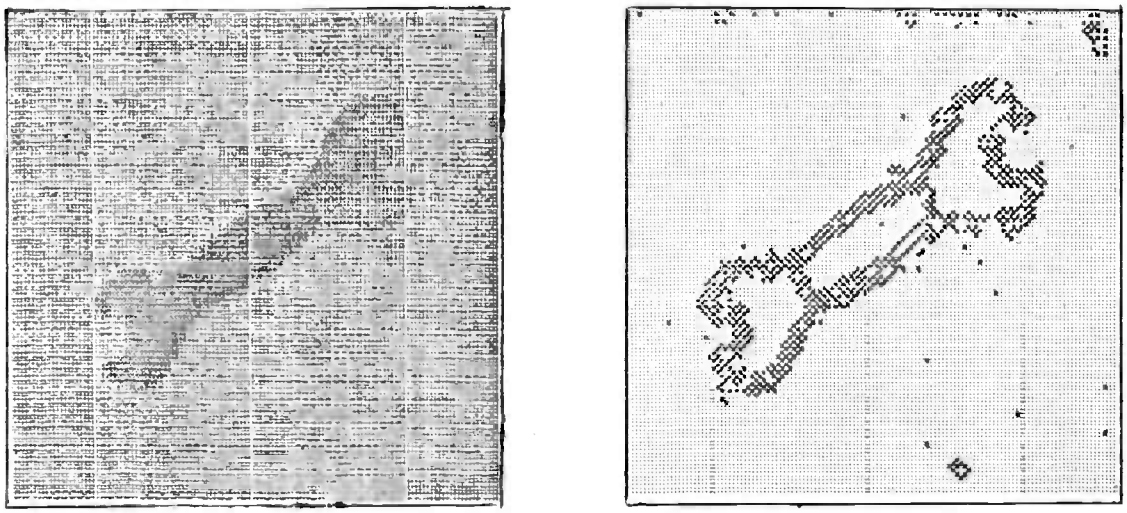
$$\|\mathbf{h} - \mathbf{h}^{(0)}\|^2 + \|\mathbf{v} - \mathbf{v}^{(0)}\|^2,$$

where

$$h_{ij} = x_{ij} - x_{i-1j} \text{ and } v_{ij} = x_{ij} - x_{ij+1}.$$

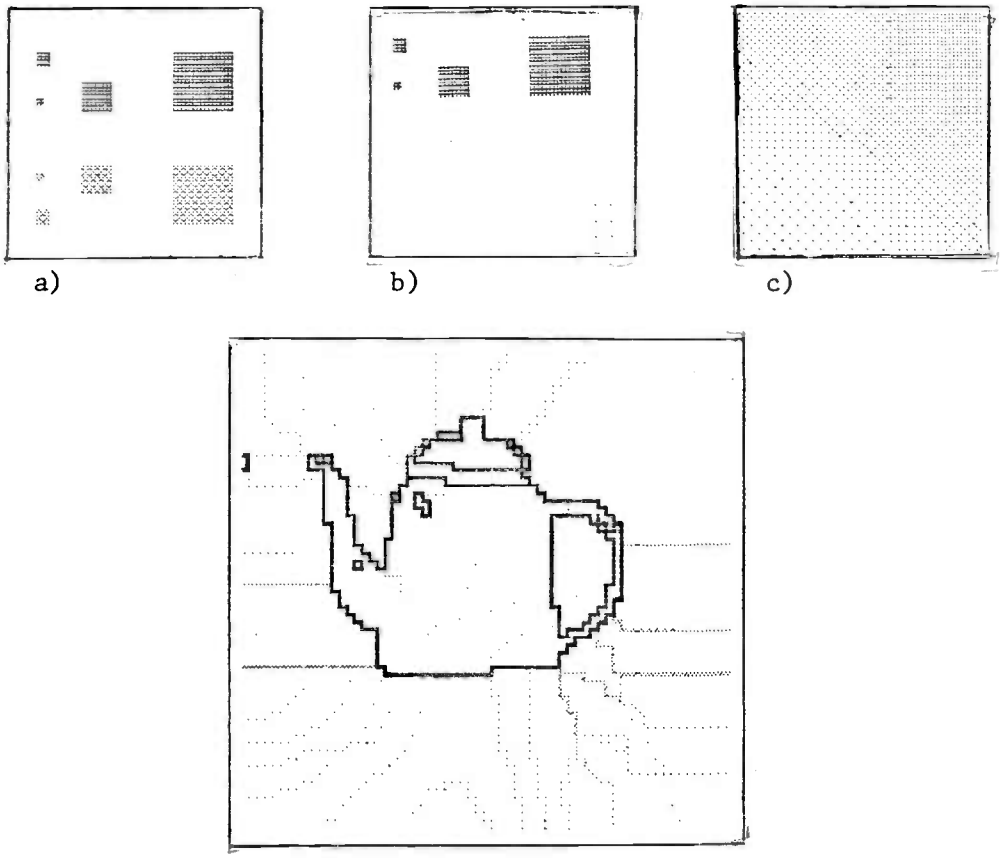
The result of using this measure in the cost function is to emphasis the importance of intensity changes in the initial data, so that the test image of squares of different sizes and contrasts, used earlier, now behaves quite differently (fig 2.16) in weak constancy relaxation. Instead of the criterion for whether the squares are levelled or retained being dependent on both their contrast and area, it now depends only on contrast.

However, on real images the results of weak constancy relaxation with the edge space measure of distance seems to be rather noisy (fig 2.16d). This may be because the least squares fitting process within each region now effectively uses data, not from the whole region as before, but just from its boundary. Using less of the data means that it is less immune to noise. Alternatively the apparently spurious lines of discontinuity may be due to heightened response to extended gradients in the image. The response is heightened because the edge space distance measure tends to preserve changes of intensity in the initial data, and occurs because the constancy constraint is inappropriate for extended gradients. If the latter explanation is correct the spurious responses would not occur when the edge space distance measure were used with the minimum energy scheme.



a) b)

Fig 2.15 Weak harmonic relaxation can cope with gradients but does not localise step edges to within one pixel as weak constancy relaxation does.



a) b) c)

d)

Fig 2.16 Weak constancy relaxation using a distance measure in edge space (gradient space): The testcard of fig 2.13, under increasing penalty constant, c, (a-c) shows that the persistence of the squares depends only on contrast, not on area. Applied to a real image (fig 1.3a) and differentiated (d): spurious intensity changes appear that are not apparent in the original image.

3 Parallel algorithms for constrained labelling: previous work.

The previous chapter discussed some uses of constrained labelling in low level vision. Such a process is often described formally in terms of affixing labels to the nodes of a graph (Waltz (1972)). Where a pair of nodes in the graph is joined by an arc, this indicates that the labels on those nodes are mutually constrained. The set of constraints characterises the labelling process by defining what labellings are acceptable as an output of that process (consistent labellings).

Specifying constraints is a declarative definition of the process. This chapter discusses methods of deriving a *procedure* for achieving a consistent labelling from that declarative definition. The declarative definition is non-deterministic in that there may be many consistent labellings. The procedure must be able to deal with this either by selecting one particular consistent labelling, or by returning a set of consistent labellings. The "principle of least commitment" (Marr (1982)) is important here: that it is undesirable to eliminate hypotheses arbitrarily, but that this must be balanced against the need to assume working hypotheses, to establish a context for further investigation. For example Waltz's program for labelling line drawings follows a cautious policy - it avoids eliminating any consistent labellings (interpretations of the drawing) - see section 3.1. Other schemes, for instance Hinton's discrete labelling algorithm, use a priori information to select a single consistent labelling. The "least disturbance" principle, used in the algorithms discussed in chapters 4 and 5, provides a natural way of making that selection when the labels are real valued and under-constrained.

A common factor in many labelling procedures is the parallelism inherent in their operation. The labelling process is distributed amongst the nodes of the graph and, in principle, processors at each nodes could work concurrently. In the case of pixel labelling processes in low-level vision, the topology of the graph is fixed by the array structure of the image data; the nodes of the graph can map directly onto the processor cells of a cellular processor array, whose processor cells act in parallel.

A distributed implementation of a labelling process is efficient provided the constraints are local - acting over small sets of nodes - so that each node is

connected to only a few others in the graph. In that case the processor at a given node need deal only with the labels on its own node and the ones on those few other nodes that are connected to it. These labelling processes are iterative. At each node labels are adjusted according to some function of the labels at that node and its neighbours. The adjustment is applied in parallel to all nodes, and is repeated until some stopping condition is reached. Although the constraints and hence the adjustment act only locally, the effect of repeated adjustment is to 'propagate' the influence of each constraint throughout the graph. That is the power of such algorithms: a repeated application of a purely local operation causes a global effect by propagation. This is particularly important for cellular array processors whose cells are interconnected only locally.

In a cellular array processor (a Single Instruction Multiple Data machine) the processors work synchronously; however the nature of labelling algorithms does not require this - the processors could quite well run asynchronously, each stopping according to its own criterion. This suggests that less rigidly structured machine architectures could be used in which a pool of processor cells serves data cells as required, as in the ALICE machine (Darlington and Reeve (1981)). This approach is appropriate not only to array structured data but also to graphs of arbitrary topology such as line drawings or semantic networks.

This chapter describes the varieties of constrained labelling schemes that have appeared in the literature and establishes a context for the algorithms described in chapters 4 and 5.

3.1 Discrete labelling

This thesis is principally concerned with determining real-valued labels subject to constraints but it is useful to look first at algorithms for discrete labelling. Algorithms for these two types of labels share some properties: they can be thought of in terms of graph labelling and are parallel and iterative, but extra difficulties arise with real-valued labels in achieving convergence (terminating the iterative algorithm correctly) and in understanding the precise computational effect.

Examples of discrete labelling problems are: labelling line drawings (Waltz's program described in chapter 2.5) and character recognition (Ullmann(1974)). In the first case the junctions of a line drawing are to be labelled consistently; the junction label specifies the geometry of the junction and the physical type of the

lines that meet there. A pair of junctions joined by a line are labelled consistently if they allow a consistent interpretation of that line. In the second case a handwritten test character is represented as an array of strokes (oriented line segments) which is to be mapped onto a reference character. Strokes in the test character are to be labelled as corresponding to strokes in the reference character, subject to the constraint that the mapping from test character to reference character be continuous.

The number of possible ways of labelling a set of nodes grows exponentially with the number of nodes, so a brute force search for consistent label assignments is infeasible. In the labelling algorithm described below, there is an upper bound on the amount of work done which is linear in the number of nodes. However, as is shown below, it does not guarantee to solve the problem completely.

Notation is introduced by Rosenfeld et al. (1976) for a labelling problem in which all the constraints involve just one node (unary constraints) or two nodes (binary constraints). Waltz's line labelling problem falls into this category. They show that there exists a greatest consistent labelling $\mathbf{L}^{(\infty)}$ and that the following algorithm always converges to this labelling. A labelling, $\mathbf{L}$, is a set.

$$\mathbf{L}=\{L_1, \ldots, L_n\}. \tag{3.1}$$

where $L_i, i=1..n$ are label sets attached to each of the n nodes of a graph. Note that this is not an assignment, to each node, of a unique label but a set of possible labels. The set of allowed labels at node i is Λ_i so that

$$L_i \subset \Lambda_i \tag{3.2}$$

- this is the unary constraint on node i. The set Λ_{ij} *for each* $i,j \varepsilon \{1,..,n\}$ specifies which pairs of labels on the two nodes are compatible - the binary constraints. For a pair of non-interacting (because unconnected) nodes

$$\Lambda_{ij}=\Lambda_i \times \Lambda_j \tag{3.3}$$

- all pairs of labels are compatible, and in practice the node pair (i,j) need never be involved in any computation Initially the labelling

$$\mathbf{L}^{(0)}=\{\Lambda_1, \cdots, \Lambda_n\}$$

is used - each node starts with all labels allowed by the unary constraints. The labelling algorithm proceeds iteratively deriving, at the m^{th} iteration,

$$\mathbf{L}^{(m+1)} \text{ from } \mathbf{L}^{(m)}=\{L_1^{(m)}, \cdots, L_n^{(m)}\} \tag{3.4}$$

by removing from each $L_r^{(m)}, r=1..n$, any label λ such that:

$$(\{\lambda\} \times L_s^{(m)}) \cap \Lambda_{rs} = \varphi \text{ for some } s \in \{1,...,n\} . \tag{3.5}$$

Any label for which there is not at least one compatible label on every other node in the graph, is removed.

It is an important property that labels can be removed in any order, according to this rule, without affecting the final result. This means that the algorithm can be executed in parallel - the labelling process can proceed independently (in parallel and asynchronously) at each node. It terminates when, at each node, none of the labels meet the condition defined in (3.5) and this must occur in a finite time because the problem itself is finite: at least one label is removed from some $L^{(m)}$ at each iteration and the $L^{(m)}, m=1..n$ contain together only a finite number of labels.

Convergence is assured but how is the resulting labelling $\mathbf{L}^{(\infty)}$ to be interpreted? It does not in general assign unique labels to each node, but a set which may contain many or just one label, or be empty. It is a 'cover' for all consistent label assignments in the sense that if $(\lambda_1, \lambda_2, \cdots \lambda_n)$ is a consistent assignment of labels to nodes 1..n, then

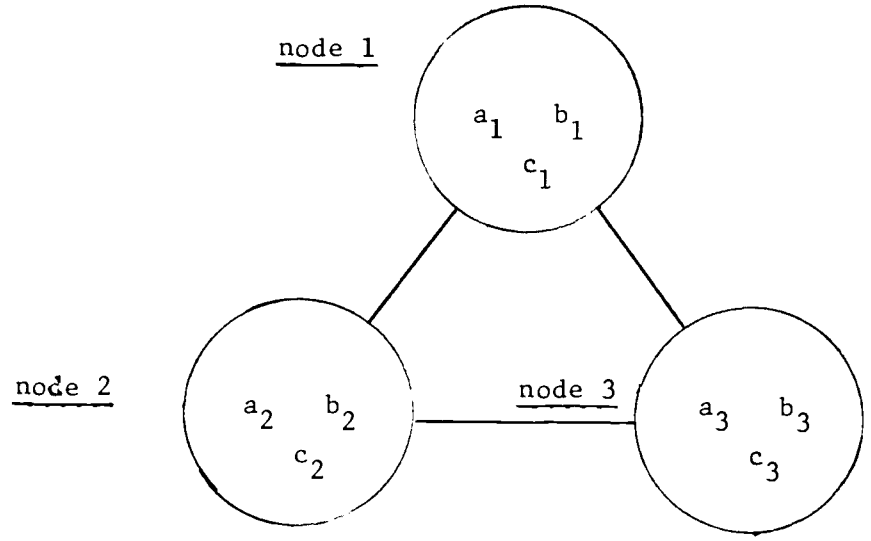
$$\forall i = 1..n \ \lambda_i \in L_i^{(\infty)}. \tag{3.6}$$

Thus if $\exists i \ s.t. L_i^{(\infty)} = \varphi$ then the problem is solved. There is no consistent assignment of labels. And if

$$\forall i \ L_i^{(\infty)} \textit{ contains exactly one label} \tag{3.7}$$

the problem is also solved - a unique consistent assignment of labels has been found. Otherwise the algorithm has determined only a cover for the set of consistent label assignments and a search must be done, under that cover, to obtain them explicitly.

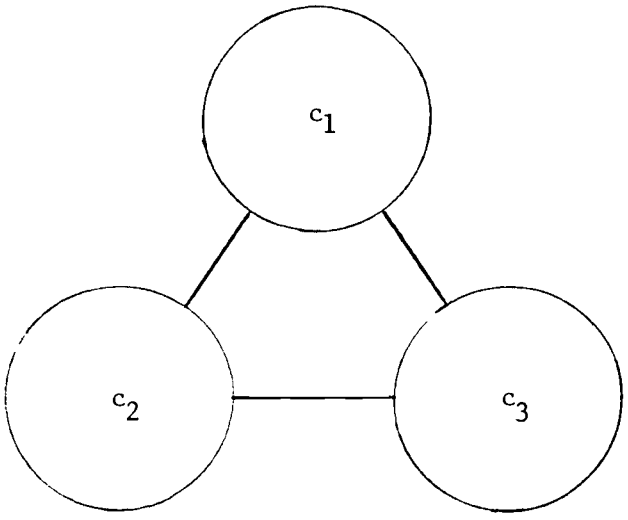
Remarkably enough, Waltz's algorithm frequently yielded a unique assignment of labels - a unique interpretation of the line drawing - as in (3.7) above. This is a remarkable result because even when only one consistent assignment of labels exists, the sets $L_i^{(\infty)}$ need not all be singletons as in (3.7) (see fig 3.1 for an example); even though the graph itself may admit only one consistent interpretation the labelling algorithm need not necessarily find it explicitly, but in Waltz's case it did. Perhaps this is a consequence of the richness of the set of constraints he used.



Compatible pairs of labels:

(a_1,a_2) (a_2,a_3) (a_3,b_1)
(b_1,b_2) (b_2,b_3) (b_3,a_1)
(c_1,c_2) (c_2,c_3) (c_3,c_1)

a)



b)

Fig 3.1 Under Waltz-type constrained labelling, a unique consistent labelling may be concealed in label sets with more than one element each. The label covers in (a) cannot be reduced any further by the Waltz filter. Nonetheless, the only consistent labelling is (b).

3.2 Probabilistic relaxation labelling

The vision problems discussed in chapter 2 included labelling edges, regions, lines and curves, and also range and surface orientation. In each case areas of an image (either single pixels or larger areas) are assigned real-valued labels subject to certain constraints. The discrete labelling algorithm just described is not appropriate to real-valued labels but perhaps it can be modified to handle them Rosenfeld et al (1976) discuss such a modification, in their case to handle probabilistic labels, which are real valued, in the range [0,1]. This is in order to be able to make estimates of the likely labelling in situations where there is uncertainty due to noisy data, or incomplete knowledge of the situation The evolution of their non-linear probabilistic algorithm is outlined in this section It has some drawbacks however. it is not guaranteed to converge, nor is it clear exactly what computation it performs. These issues are clarified in the optimisation based algorithms presented in the next two chapters.

3.2.1 Fuzzy relaxation

Fuzzy logic (Zadeh (1965)) can be used to convert the discrete relaxation scheme into a probabilistic one (the labels are therefore real-valued), in a simple way. The word probabilistic is used loosely to refer to some measure of confidence, lying in the range [0,1]. The labelling at a node, instead of being represented by a boolean valued function of the possible labels, is represented by a real-valued membership function, with range [0,1]

In the fuzzy case the labelling at a node, L_i, becomes an ordered set of real valued functions In the notation of Rosenfeld et al.

$$\mathbf{L}=\{L_1, \cdots, L_n\} \text{ with } L_i:\Lambda\rightarrow[0,1],\ i=1..n \tag{3.8}$$

where Λ is the set of all possible labels. Similarly there are 'a priori' labellings

$$\Lambda_i : \Lambda\rightarrow[0,1] \tag{3.9}$$

and compatibilities

$$\Lambda_{ij} : \Lambda_i\times\Lambda_j\rightarrow[0,1]. \tag{3.10}$$

By analogy with the discrete case (3 3), for unconnected nodes i,j we have

$$\Lambda_{ij}(\lambda,\mu)=1\ \forall\lambda,\mu\varepsilon\Lambda. \tag{3.11}$$

In the discrete case a label assignment $\{\lambda_1, .,\lambda_n\}$ is consistent when

$$\forall ij \quad (\lambda_i,\lambda_j)\varepsilon\Lambda_{ij} \tag{3.12}$$

and in the fuzzy case this condition becomes

$$\forall ij \quad \forall\lambda,\mu\varepsilon\Lambda,\ \Lambda_i(\lambda)\leq\Lambda_{ij}(\lambda,\mu) \text{ and } \Lambda_j(\mu)\leq\Lambda_{ij}(\lambda,\mu). \tag{3.13}$$

As in the discrete case the algorithm operates on an initial labelling

$\mathbf{L}^{(0)}=\{\Lambda_1, \ldots, \Lambda_n\}$ and derives

$\mathbf{L}^{(m+1)}$ from $\mathbf{L}^{(m)}=\{L_1^{(m)}, \ldots, L_n^{(m)}\}$ as follows:

for each i, $\lambda\varepsilon\Lambda$

$$L_i^{(m+1)}(\lambda)=min(L_i^{(m)}(\lambda),Q_i^{(m)}(\lambda)) \tag{3.14}$$

where

$$Q_i^{(m)}(\lambda)=\inf_{j\neq i}\left(\sup_{\lambda'\varepsilon\Lambda}(min(L_j^{(s)}(\lambda'),\Lambda_{ij}(\lambda,\lambda')))\right).$$

The use of inf and sup derives from their equivalence, in fuzzy logic, to set union and intersection.

This updating formula has the effect of progressively reducing label probabilities until they satisfy a condition for consistency, that

$$\forall i,j\varepsilon\{1,..,n\},\ \forall\lambda\varepsilon\Lambda,\ \exists\lambda'\varepsilon\Lambda\ s.t. \tag{3.15}$$

$$L_i(\lambda)\leq min(L_j(\lambda'),\Lambda_{ij}(\lambda,\lambda')).$$

The label strengths are monotonic decreasing so it is also clear that the algorithm must converge. In fact the fuzzy logic derivation guarantees convergence: any fuzzification of a convergent boolean scheme must also be convergent (see appendix A.). Moreover the proof shows that convergence must occur within a finite number of iterations, not just as the limit of $\mathbf{L}^{(m)}$ as $m\to\infty$.

Just as, in the discrete labelling, labels are removed from but never replaced in the sets $L_i^{(m)}$, in the fuzzy scheme the $L_i^{(m)}(\lambda)$ are monotonically decreasing in m This is entirely appropriate because, again by analogy with the discrete case, the $L_i^{(m)}(\lambda)$ is a label cover - a set of upper bounds on the fuzzy membership values for each label on each node.

3.2.2 Linear probabilistic relaxation

In the fuzzy scheme described above the value $L_i^{(m)}(\lambda)$ was an upper bound to the membership value for label λ on node i. Inequality constraints place upper bounds on the membership values, and are propagated to decrease iteratively the values of the $L_i^{(m)}(\lambda)$. Frequently, when dealing with probabilities, it is useful also to have lower bounds on the membership values as used in the INFERNO system (Quinlan (1982)). These are then iteratively raised, as dictated by constraints To see that it is useful to maintain lower bounds, consider the simple case of two mutually exclusive and exhaustive labels λ,μ, on a single node They are subject to the constraint on their probabilities that:

$$p(\lambda)+p(\mu)=1. \tag{3.16}$$

Any upper bound on $p(\lambda)$ automatically, therefore, implies a lower bound on $p(\mu)$.

Maintaining lower bounds in addition to upper bounds provides a mechanism by which constraints can propagate to support a given label, as well as inhibiting it as in the fuzzy scheme.

Rosenfeld et al. (1976) attempted to implement a scheme which allows both positive and negative support to be given to a label, under a set of equality constraints. In place of $L_i(\lambda)$ they refer to a probability $p_i(\lambda)$, for label λ on node i. Instead of Λ_{ij} Rosenfeld et al suggest the use of the following set of constraints·

$$\forall i, 1 \leqslant i \leqslant n,\ \forall \lambda \varepsilon \Lambda,\ p_i(\lambda)=\sum_{j=1}^{n}\left(\sum_{\mu} s_{ij}(\lambda,\mu)p_j(\mu)\right) \tag{3.17}$$

where the matrix of coefficients

$$s_{ij}(\lambda,\mu)=c_{ij}p_{ij}(\lambda,\mu) \tag{3.18}$$

and p must satisfy

$$p_{ii}(\lambda,\mu)=\begin{matrix}1 \text{ if } \lambda=\mu \\ 0 \text{ otherwise}\end{matrix}. \tag{3.19}$$

The weights c_{ij} are introduced to adjust the relative contribution from each node in the neighbourhood of i, including i itself. The quantities c_{ij} and $p_{ij}(\lambda,\mu)$ are all positive so that $s_{ij}(\lambda,\mu)$ is also positive. This constraint on $s_{ij}(\lambda,\mu)$ is applied to ensure that the $p_i^{(n)}$ remain positive, for all n. Linear algebra theory shows that the following iterative formula yields, in the limit as $n \to \infty$, a consistent labelling - one that satisfies the constraints in (3.17). From the consistency condition, an updating formula is derived·

$$p_i^{(n+1)}(\lambda)=\sum_{j=1}^{n}\left(\sum_{\mu}s_{ij}(\lambda,\mu)p_j^{(n)}(\mu)\right). \tag{3.20}$$

Since this updating formula acts in parallel over the nodes of the graph, and is linear, its convergence can be analysed by conventional linear matrix algebra. To maintain

$$\forall i \ \sum_{\lambda}p_i^{(n)}(\lambda)=1 \tag{3.21}$$

(on each node, probabilities of labels sum to one) the constraints are imposed that:

$$\forall i,j \ \sum_{\lambda}p_{ij}(\lambda,\lambda')=1 \ \textit{and} \ \ \forall i \ \sum_{j}c_{ij}=1. \tag{3.22}$$

However, it is also shown that under certain conditions the final set of converged probabilities $p_i^{(\infty)}(\lambda)$ is entirely independent of the original set $p_i^{(0)}(\lambda)$. It is clear that this is unacceptable if the relaxation scheme is regarded as a process with $\{p_i^{(0)}\}$ as input and $\{p_i^{(\infty)}\}$ as output - the output is independent of the input. Instead of combining information in the initial labelling (prior probabilities) with the constraints, information is drawn from the constraints only. This is not surprising, however, when we consider the constraints (3.17). There are nl constraints on nl unknowns (n nodes with l labels on each). The unknowns could be fully constrained. Whether or not this is so can be determined from the fixed points of the iterative scheme. Any fixed point $\mathbf{p}$ (the vector whose components are $p_i(\lambda)$) satisfies

$$\mathbf{p} = S.\mathbf{p}, \tag{3.23}$$

which implies

$$(S-I).\mathbf{p}=0. \tag{3.24}$$

The dimensionality of the space of fixed points is therefore

$$nl-rank\,(S-I), \tag{3.25}$$

which in turn depends on the multiplicity of the unity eigenvalue of S. (It can be shown that S does have a unity eigenvalue.) In the general case the multiplicity need not be greater than one, so the space of fixed points is one-dimensional. Therefore, in general, the constraints in (3.17), together with the normalisation condition (3.21) fully constrain the result $\mathbf{p}^{(\infty)}$.

The conclusion is that the constraints in (3.17) are inappropriate because they are too strong.

3.2.3 Non-linear probabilistic relaxation

Because they considered linear relaxation to be unsatisfactory, Rosenfeld et al. proposed a non-linear formula for updating the $\mathbf{p}^{(n)}$, of the form †:

$$p_i^{(n+1)}(\lambda)=\frac{Q_i^{(n)}(\lambda)p_i^{(n)}(\lambda)}{\sum_{\beta\in\Lambda} Q_i^{(n)}(\beta)p_i^{(n)}(\beta)} \tag{3.26}$$

where

$$Q_i^{(n)}(\alpha)=\sum_{j\beta} t_{ij}(\alpha,\beta)p_j^{(n)}(\beta)$$

for some matrix T.

A labelling of the form:

$$\forall i,\lambda\ \exists\lambda'\ s.t.\ p_i(\lambda)=\begin{matrix}1 & when\ \lambda=\lambda'\\ 0 & otherwise\end{matrix} \tag{3.27}$$

- known as an unambiguous labelling - may be a fixed point of the formula (3.26) and stable subject to certain conditions (appendix B.1). There is another type of fixed point under (3.26) occurring when:

$$\forall i\ \sum_{j\beta} t_{ij}(\alpha,\beta)p_j(\beta)=C_i\ -\ a\ constant \tag{3.28}$$

Equation (3.28) represents nl linear constraints, on nl variables, (there are n nodes, l possible labels for each) and there are n degrees of freedom for the choice of the C_i. In general, there is an n dimensional space of these fixed points. They are generally unstable - see appendix B. for proof and conditions for stability. Convergence to such a fixed point is not entirely precluded but will only occur if the point is approached from certain directions in label space. Peleg (1980) shows that in a similar non-linear probabilistic scheme, the a priori labelling

$$p_i(\lambda)=prior\ prob(\ label\ \lambda\ on\ node\ i)$$

is one such fixed point

† The formula given here is equivalent to that of Rosenfeld et al. but expressed slightly differently.

The non-linear probabilistic scheme has major disadvantages: there is no guarantee of convergence and in any case it is not clear what computation is being performed. There have been attempts to analyse convergence. Zucker et al. (1981) derive a condition under which convergence is assured. The final labelling in that case is unambiguous and stable, of the type described in appendix B.1. If the condition holds at the n^{th} iteration, for some n, the label on each node with the highest probability is adopted as the final label for that node (it has probability 1 in the final labelling). Hence, under this condition, the scheme degenerates to a simple process of selecting the most probable label on each node. Haralick et al. (1980) analyse the effect of instability of an unambiguous fixed point in one particular labelling problem. However no general results on stability of the non-linear relaxation scheme are available. Hanson and Riseman's edge relaxation algorithm, mentioned in the last chapter, is of the same basic type. Although its effect appears to include some enhancement of edges, it does not always converge (fig 2.6d) - oscillation can occur.

The spirit of non-linear relaxation is to represent co-operation and competition in a network in which labels support and inhibit one another. Similar models have been proposed for networks of interacting neurons (Sejnowski (1981)) and for a processing network for random dot stereograms (Marr (1978)) and here the stability of the network is difficult to analyse (Marr et al. (1977)) and fragile in practice (Mayhew (1982)). Oscillatory systems of this general kind are observed in iterative application of certain operators to boolean arrays. In some cases a concise condition for stability can be found (e.g. Blake (1982)) but even apparently simple cases, such as repetitive application of the the median filter, are difficult to analyse (Lester et al. (1980)).

This thesis will go on to propose the use of co-operation and competition, both linear and non-linear, but uses the paradigm of constrained optimisation to ensure convergence Moreover the effect of the computation is clear - to satisfy explicitly stated constraints.

3.3 Constrained labelling by optimisation.

A paper by Ullman (1979) sets out a formal framework in which optimisation under local inequality constraints is achieved by a local iterative process. Real valued labels on the nodes of a graph become variables $\{x_i, i=1.\ n\}$ in an optimisation problem and are mutually constrained. In addition an "objective function"

is used that measures the "cost" of a given labelling. This cost is to be minimised, subject to the constraints.

As in the labelling schemes described so far, each constraint is local in the sense that it constrains only a few nodes: if the constraints are

$$g_j(\mathbf{x}) \geq 0,\ j = 1..n \tag{3.29}$$

then for each j,k, $\partial g_j / \partial x_k$ depends only on "a few" of the x_i. Similarly the cost function f is a local function of $\{x_i\}$ in that, for each k, $\partial f / \partial x_k$ depends only on a few x_i. This means that the iterative process used to perform the optimisation can be decomposed, as were the schemes already described, into a number of local processes that can be executed in parallel. The iterative process is, as before, essentially a discrete time model of cooperation and competition in a network. The interaction between nodes may either be linear or non-linear, depending on the precise nature of the objective function, f, and the constraints (3.29).

The iterative formula is essentially a hill-climbing process and is guaranteed to converge. Moreover the purpose of the computation itself is clearly set out. to achieve the solution of the optimisation problem It forms, therefore, a firm basis on which to proceed with the vision tasks discussed in this thesis

The final task in this chapter is to review some of the reported work on constrained labelling by optimisation, in order to introduce the new work described in the next chapter.

3.3.1 Real valued labelling

Some of the most successful work using relaxation by optimisation is the shape from shading algorithm of Ikeuchi and Horn (1980) and the algorithm for deriving optical flow by Horn and Schunk (1980). They use essentially similar formal schemes for relaxation so it suffices to describe the first. The problem is to label each pixel in an image with the local surface orientation at the corresponding point in the scene, given only the image intensity and a model of the illumination of the object. Surface orientation is a normalised vector and may be represented by a pair of direction cosines (p_{ij}, q_{ij}) at the image point (i,j), whereas the image intensity I_{ij} has only one value at each point. Hence the equation

$$\forall ij \ \ R(p_{ij}, q_{ij}) = I_{ij} \tag{3.30}$$

($R(p,q)$ is the intensity predicted by the illumination model for a surface at orientation (p,q)) does not fully constrain (p_{ij},q_{ij}). Continuity of the surface is invoked to determine the (p_{ij},q_{ij}) fully. Continuity here cannot be regarded as an equality constraint like (3.30) because continuity is not well defined (as was discussed in chapter 2) for sampled data - that is for a function over a discrete domain. The approach taken by Ikeuchi and Horn is to minimise a measure of lack of smoothness:

$$F = \sum_{ij} [D(p_{ij})]^2+[D(q_{ij})]^2 \tag{3.31}$$

where

$$D(x_{ij}) = x_{i+1,j}+x_{i-1,j}+x_{i,j-1}+x_{i,j+1}-4x_{ij}. \tag{3.32}$$

A logical strategy would now seem to be to minimise (3 31) subject to the constraints (3.30) to choose the smoothest consistent labelling. However, Ikeuchi and Horn do instead an unconstrained minimisation of the quantity:

$$E=\sum_{ij}(R(p_{ij},q_{ij})-I_{ij})^2 + \lambda F, \tag{3.33}$$

where F is defined in (3 31). The effect is to trade the accuracy with which (3.30) holds against the smoothness of the labelling; the precise balance of the trade-off is determined by the constant λ which sets the relative weights given to the two components in (3.33). Treating (3.30) as an equality constraint would be equivalent to setting λ to some very small value. Horn and Schunk (1980) suggest, however, that λ should in practice be set large enough to provide some immunity from digitisation and sensor noise in the intensity I. The minimisation of (3 33) is achieved essentially by hill-climbing - following an ascending path over E in (3 33). This results in a local, iterative algorithm, in which the repeated local action achieves a global effect.

Burch et al. (1983) describe an algorithm for maximum entropy restoration of degraded images. The restoration is similarly expressed as a constrained optimisation. to maximise entropy over the image subject to constraints that the final image be consistent with the degraded data. They use a development of the basic hill climbing idea to achieve this, but improved for faster convergence. Similar optimisation approaches have been used by Narayanan et al. (1982) and for probabilistic labelling by Berthod(1982), Faugeras and Berthod (1981). The scheme of Narayanan et al is basically similar to that of Ikeuchi and Horn: it fits intensity data to a smooth distribution, by minimising the sum of a distance measure and a roughness measure. It is important to realise that such a

process does *not* in any way achieve a labelling of discontinuities in intensity. It simply performs smoothing by optimising a convex objective function. This is not surprising as it was argued in chapter 2 that an objective function for labelling discontinuities must be non-convex. A smoothing operation simply distributes roughness (as defined by some measure such as (3.31)) evenly, whereas what is required is to localise it in discontinuities whose spatial extent is limited. Smoothing is inadequate because it does not satisfy the necessary weak constraint of continuity "almost everywhere"; instead, the result of smoothing is "almost continuous" everywhere. Realising the deficiencies of the smoothing operation, Narayanan et al. attempt to make an initial step towards a segmentation scheme, that labels foreground and background in an image. Even in this relatively simple task, the cost function becomes non-convex and therefore hill-climbing alone cannot reach the global optimum.

Berthod constructs a cost function like (3.33) above, which is the sum of two components: one measuring the displacement of the labelling from its initial state and another measuring the 'inconsistency' of the labelling. The cost is then minimised by relaxation to reach a compromise between conserving the initial labelling and making the labelling more consistent. As in the shape from shading algorithm of Ikeuchi and Horn above, there is no set of constraints which hold exactly; in fact Faugeras and Berthod refer to the same constraints as in chapter 3.2.2 (3.17) which, if they hold exactly, over-constrain the problem. Instead they are made to hold as closely as possible, by minimising an inconsistency measure. While recognising that the measure is non-convex and seeking its global optimum, Berthod does not suggest a method of finding the global optimum, other than reaching a local optimum by hill climbing.

In the chapter 5 we discuss optimisation under linear constraints in an algorithm quite similar to that of Ikeuchi and Horn. However, for this purpose it is appropriate to use equality constraints, which are maintained exactly (as when λ is very small in the scheme above). The constraints define a model of what constitutes an acceptable labelling; the algorithm finds the best fit to that model by minimising the displacement from an initial labelling subject to the constraints. A further development of this idea is the weak constraint - an equality constraint which may be broken occasionally - used to represent the notion of "continuity almost everywhere". The hill-climbing algorithm can be most conveniently extended to deal with explicit constraints by using Lagrange multipliers, either with inequality constraints (Ullman,S. (1979)) or with equality constraints (see chapter 4.1). These ideas are fully discussed in chapter 4.

3.3.2 Discrete labelling

Another successful use of an iterative optimisation algorithm for labelling is Hinton's (1978) puppet recogniser. This is essentially a system for reasoning, in which hypotheses interact via logical constraints. The purpose of the scheme is to produce a true/false labelling for the hypotheses which satisfies the constraints. (In the case of the puppet, modelled as an assembly of rectangles, the hypotheses concern the presence or absence of, for instance, a rectangle to represent head. A typical constraint might require the head rectangle to be adjacent to the neck rectangle.) In practice there may be a number of consistent labellings (just as in Waltz's scheme, chapter 3.1) so each hypothesis is tagged with a preference P_i and the 'best' of the consistent labellings is then chosen - the one that maximises

$$\sum_{i=1}^{n} P_i S_i$$

where

$$S_i = \begin{matrix} 1 \text{ if the } i^{th} \text{ hypothesis is true} \\ 0 \text{ otherwise} \end{matrix}$$

This is an integer programming problem and not, as it stands, susceptible to continuous relaxation methods. In fact it is an extreme case of a non-convex programming problem - labelling space is not merely non-convex, it is not even connected

Hinton converts the integer problem to a continuous one by allowing the S_i to vary in the interval [0,1] and constructing a convex set around the feasible points in label space (the points corresponding to consistent labellings) in the integer problem. This transformed continuous optimisation problem yields a solution for the S_i. If $\{S_i\}$ is the solution of the continuous optimisation problem and $\forall i\ S_i=0$ *or* 1 then a fortiori $\{S_i\}$ is the solution to the integer problem. If the S_i are not all integers the convex set must be made smaller - fitting "more tightly" around the feasible points - to force an integer solution. But it is far from obvious in the general case, how to do this. The integer solution is guaranteed if the convex set is the *convex hull* of the feasible points (the smallest convex set that contains them). But in general the construction of a convex hull in a large dimensional space would be expensive.

In practice Hinton found that an integer solution was achieved without any need to modify the convex set first constructed. Whether this result has general applicability is a matter for speculation: it may be that it applies when the

problem is tightly constrained as in Waltz's line labelling (chapter 3.1) in which a rich set of constraints similarly led to a surprisingly conclusive labelling. A somewhat analogous strategy is adopted in this thesis (chapter 4) for dealing with non-convex continuity constraints (weak constraints) in continuous relaxation. Here, however, the labelling is by no means entirely determined by the constraints and the convex set strategy does not always, on its own, yield a solution. It is nonetheless very useful when followed by a further computation, and this is fully dealt with in chapter 4.

4 Parallel algorithms for imposing weak continuity constraints

In chapter 2 "weak continuity" in an array of data was defined in terms of constraints which can be broken at a cost - a fixed penalty is paid for each broken constraint. The sum of these penalties over the array is then a simple global measure of total "degree of discontinuity" in the array Given some initial data array, discontinuities in the data are labelled by forming a new array which minimises the sum of:

difference between the new array and initial one (measured by euclidean distance)

PLUS

the sum of penalties for broken constraints.

The positions of broken constraints in the new array mark discontinuities.

The simplest form of continuity is constancy; the effect of imposing weak constancy constraints on image intensity data and on angle data was discussed in chapter 2. This chapter explains, in detail, how the cost minimisation problem is solved. First the theoretical basis of the "graduated non-convexity" algorithm is set out. Then its implementation is described, both for a parallel array processor and for an asynchronous parallel machine. The results of computer simulations of the algorithm, on both types of architecture, are discussed.

4.1 Optimisation under weak constancy constraints

In chapter 2 we saw that weak constancy constraints can be used to label discontinuities in intensity data to produce a line drawing, and in angle data to mark vertices between lines. The constraints are imposed with least disturbance to the initial data array $\mathbf{x}^{(0)}$, to produce a new array $\mathbf{x}$. The array x_{ij} is the solution to an optimisation problem (eq 2.24) to minimise

$$F(\mathbf{x}) = \sum_{ij} \left(\|x_{ij} - x_{ij}^{(0)}\|^2 + c^2 \lceil x_{ij} \neq x_{i,j+1} \rceil + c^2 \lceil x_{ij} \neq x_{i-1,j} \rceil \right) \qquad (4.1)$$

$$\text{where } \lceil b \rceil = \begin{cases} 1 & \text{if } b \text{ is true} \\ 0 & \text{otherwise} \end{cases}.$$

We showed (in section 2.4.1) that this function is non-convex and therefore the

problem is generally unlikely to be susceptible to exact solution in a reasonable time. Instead, it is proposed to use the method of graduated non-convexity to achieve a good suboptimal solution, in a manner that allows parallel computation.

What methods are available, within mathematical theory, for solving this sort of problem? Considering the very substantial body of literature on optimisation there is relatively little on non-convex optimisation. Ritter's method for non-convex programming (Cottle and Mylander (1970)) is an exact method but applies only to optimising quadratic functions. The function, F, in (4.1) is not wholly quadratic so this method is not applicable. Another candidate method , which is attractive because F is a sum of local functions, is dynamic programming (Bellman and Dreyfus (1962)). Dynamic programming has been used to construct boundaries in images, using optimal line following algorithms (Montanari (1971)) but is not a parallel method as it involves sequential search. Moreover in a two-dimensional array, there is no unique natural order of the array elements; this has the consequence that to minimise even a local function like F in (4.1) involves interaction between large numbers of array elements and requires impractically large look-up tables. Ullmann(1982) has proposed a method of pruning the search in dynamic programming, using discrete relaxation. This could be efficiently implemented on a suitable parallel architecture but, in general, some search is still needed to find a global optimum. In weak continuity relaxation, search would proceed serially over the N^2 variables in the image array. As N^2 is large for real images, such a search is unlikely to be feasible. The method of graduated non-convexity, proposed in this thesis, avoids search but at the expense of finding only a good suboptimal solution, rather than the true global optimum.

Another method that makes use of the local property of F is hill-climbing by relaxation, which was discussed in chapter 3 (section 3.3.1). Where the cost function is non-convex and a global optimum is required, both Ullman (1979) and Berthod (1982) suggest settling for a local optimum, as being sub-optimal but readily attainable. This is inapplicable in our case because the initial labelling, $\mathbf{x}^{(0)}$, is already a local minimum of F. This is apparent by inspection: clearly $\mathbf{x}^{(0)}$ is a local minimum of the first term in (4.1) and the effect of small changes of x_{ij} around $\mathbf{x}^{(0)}$, for some (i,j), can only be to increase the second term (if $x_{ij}^{(0)}=x_{i,j+1}^{(0)}$) or else leave it unchanged (provided the change in x_{ij} is small enough). Similarly the third term can only increase or stay the same. No small change in $\mathbf{x}$ around $\mathbf{x}^{(0)}$ can reduce $F(\mathbf{x})$ and therefore hill-climbing from the

initial labelling has no effect - it leaves the initial state unaltered.

A non-deterministic, statistical algorithm for global optimisation has been used in thermodynamics (Metropolis et al (1953)). Hinton and Sejnowski (1983) have investigated this algorithm for use in a model of perception and learning and showed that it can be executed in parallel. It works by allowing random changes in $\mathbf{x}$ which may either increase or decrease F, but incorporating a certain statistical bias in favour of the latter. By gradually increasing this bias (lowering the "temperature" of the system) a stable state is reached. To reach a state that is close to optimal the temperature must be reduced slowly ("annealing", from the thermodynamic point of view) and this would result in long execution times. However the statistical method is potentially relevant and deserves investigation in this application.

The method of graduated non-convexity is quite different from the thermodynamic method, in that it is not statistical and makes use of the structure of the particular objective function, F, that occurs under weak constraints, to efficiently achieve a good sub-optimal solution. The graduated non-convexity method consists of two stages: first convex envelopes are used to construct a convex function F^* and the global minimum of F^* is found. Then hill-climbing is done on each in turn of a sequence of non-convex functions between F^* and F - this repeated hill-climbing is rather like trying to keep a rubber dinghy afloat on the crest of a moving wave. The result is a labelling $\mathbf{x}$ that is likely to be close to optimal - its cost $F(\mathbf{x})$ is close to the global optimum of F.

4.1.1 Convex envelopes

We now investigate the effect, with weak constancy constraints, of substituting for the cost function F a particular convex function F^*. To describe how such a function can be constructed we first consider a highly simplified problem with only two variables, x and y, in which the cost function is $F(x,y)$:

$$F(x,y) = (x-x^{(0)})^2+(y-y^{(0)})^2+c^2[x \neq y]. \tag{4.2}$$

This can be rewritten as

$$F(x,y) = \tfrac{1}{2}(x+y-x^{(0)}-y^{(0)})^2 + f\left[\frac{x-y}{\sqrt{2}}, \frac{x^{(0)}-y^{(0)}}{\sqrt{2}}\right]. \tag{4.3}$$

where

$$f(u,u_0) = (u-u_0)^2+c^2[u \neq 0]. \tag{4.4}$$

The non-convexity of F is due to f. Figure 4.1a shows a sketch of f. The function F^* is constructed by replacing f, which is non-convex in u, by its convex envelope f^*, shown in fig 4.1b, which is:

$$f^*(u,u_0)= \begin{matrix} 2(c\,|u\,|-u_0u)+u_0^2 \text{ if } |u|<c \\ c^2+(u-u_0)^2 \text{ otherwise} \end{matrix} . \tag{4.5}$$

It is straightforward to show that f^* is the convex envelope of f. Also F^*, in this simple two-variable case, is the convex envelope of F although, as we shall see, this property does not generalise to large numbers of variables. What this property means, in this two-variable case, is that

$$F^* \varepsilon S \tag{4.6}$$

where the class of functions, S, is defined to be:

$$S= \{G:\ \forall x,y\ \ G(x,y)\leq F(x,y) \text{ and } G \text{ is convex} \} \tag{4.7}$$

and that it is the maximal such function, that is:

$$\forall G \varepsilon S\ \ \forall x,y\ \ F^*(x,y)\geq G(x,y). \tag{4.8}$$

Any function and its convex envelope share the same global minimum (this can be seen to be true of the functions in fig 4.1). Therefore optimising F^* is equivalent to optimising F, but the optimisation can be achieved by hill climbing because F^* is convex. The two variable problem is therefore completely solved by this method of convex envelope construction We are interested, however, in a cost function of many variables (in the case of an image, one variable per pixel) and the convex envelope construction can be generalised in a way that *sometimes* yields the correct solution, as will now be explained.

In order to generalise to a problem with many variables, it will be convenient to deal with functions g, g^*, defined to be:

$$g(u)= f(u,u_0)-(u-u_0)^2= c^2[u\neq 0] \tag{4.9}$$

and

$$g^*(u)= f^*(u,u_0)-(u-u_0)^2 = \begin{cases} 2c\,|u\,|-u^2 \text{ if } |u|<c \\ c^2 \text{ otherwise} \end{cases} . \tag{4.10}$$

Then F in (4.2) can be rewritten as

$$F(x,y)= (x-x^{(0)})^2+(y-y^{(0)})^2+g\left[\frac{x-y}{\sqrt{2}}\right] \tag{4.11}$$

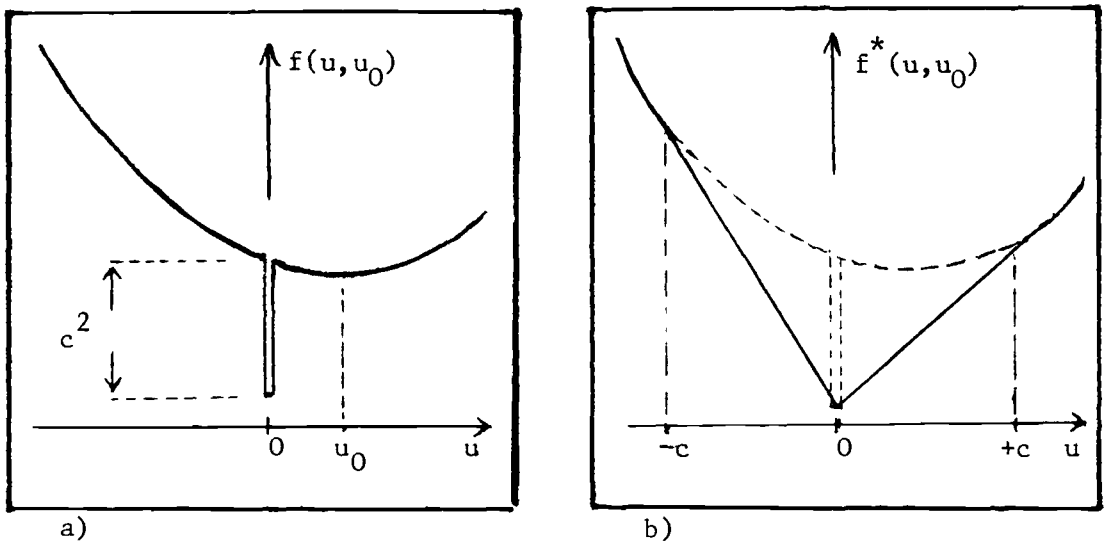


Fig 4.1 The simplest case of weak constancy relaxation - a two element array, in which the values of elements are subject to a weak constancy constraint. The difference between the values is u, initially u_0. a) Cost $f(u,u_0)$ as a function of u.
b) f^* is the convex hull of f and clearly has the same global minimum.

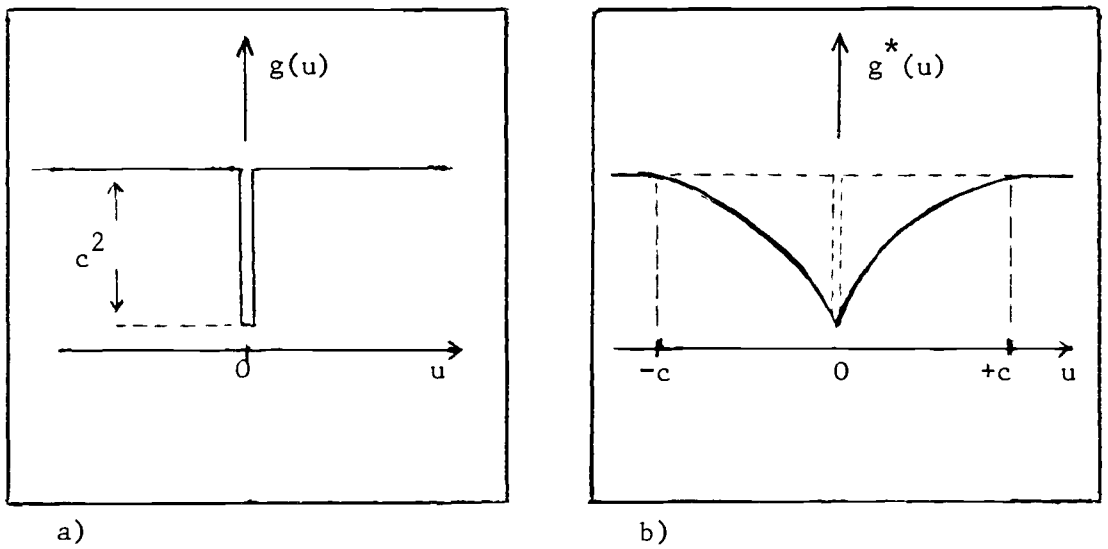


Fig 4.2 Weak constancy on arrays of many elements. It is useful to consider the component of the neighbour interaction due to the weak constancy constraint. a) Function g is f, less its quadratic component, leaving just the cost of breaking the constancy constraint.
b) Similarly, g^* is f^* less that same quadratic component.

and F^* in (4.3) is written as

$$F^*(x,y)=(x-x^{(0)})^2+(y-y^{(0)})^2+g^*\left(\frac{x-y}{\sqrt{2}}\right). \tag{4.12}$$

The form for F, in (4.11), makes it quite clear that the interaction between neighbours x and y is completely described by the function g, and similarly for g^* in F^*. Both g and g^* are illustrated in fig 4.2.

The natural generalisation for the multivariable problem is now to define:

$$F(\mathbf{x})=\sum\left[x_{ij}-x_{ij}^{(0)}\right]^2+\sum g\left(\frac{x_{ij}-x_{i,j+1}}{a}\right)+\sum g\left(\frac{x_{ij}-x_{i-1,j}}{a}\right) \tag{4.13}$$

$$F^*(\mathbf{x})=\sum\left[x_{ij}-x_{ij}^{(0)}\right]^2+\sum g^*\left(\frac{x_{ij}-x_{i,j+1}}{a}\right)+\sum g^*\left(\frac{x_{ij}-x_{i-1,j}}{a}\right), \tag{4.14}$$

where a is some constant, to be determined. In fact it is set to $a=\sqrt{8}$, chosen to ensure that F^* is convex, while being as close as possible to F (see appendix D.4). However, although F is convex it is not, in the multivariable case, the convex envelope of F^*. Therefore optimisation of F^* (which can be done by hill climbing, as F^* is convex) does not necessarily yield the global optimum of F - that would be far too easy. What can be said is that if $\mathbf{x}^*$ is the global minimum of F^* then

$$F^*(\mathbf{x}^*)=F(\mathbf{x}^*) \tag{4.15}$$

is a sufficient condition that $\mathbf{x}^*$ is also the global minimum of F. The proof is straightforward. Assume that condition (4.15) holds. First, by inspection of (4.9) and (4.10), or alternatively by inspection of fig 4.2,

$$\forall u\ g^*(u)\le g(u), \tag{4.16}$$

and in turn, from (4.13) and (4.14):

$$\forall \mathbf{x}\ F^*(\mathbf{x})\le F(\mathbf{x}). \tag{4.17}$$

As x^* is the global minimum of F^*

$$\forall \mathbf{x}\ F^*(\mathbf{x}^*)\le F(\mathbf{x}) \tag{4.18}$$

and therefore from (4.15) it follows that

$$\forall \mathbf{x}\ F(\mathbf{x}^*)\le F(x) \tag{4.19}$$

- $\mathbf{x}^*$ is the global minimum of F.

Optimising F^* is a method of optimising F that sometimes works and it has been demonstrated that there is a condition which, if satisfied, indicates that the method has succeeded. This is analogous to the situation in Hinton's method (see section 3.3.2) for optimisation over a non-convex set, done by replacing the set with its convex hull. In his case it transpired that the method very often succeeded, although he did begin to discuss a strategy for recovery when it fails. In our case however, the method frequently fails and this is demonstrated for the simple case of a step discontinuity in a linear array, in fig 4.3: the method succeeds when the penalty, c^2, for breaking a constraint is either sufficiently large or sufficiently small. In such cases the minimum cost state has very much lower cost than alternative states. But for intermediate values of c the minimum cost state is less marked and the method fails, producing a solution that falls between the initial state and the minimum cost state. In a picture array the result is that the method succeeds in some areas of the array and fails in others (see fig 4.4). Analysis of how this method works for a step discontinuity in a linear array is given in appendix D.2.

Clearly, what is needed now is a good strategy for recovery on those occasions when the convex envelope method fails.

4.1.2 Graduated non-convexity

The simplest recovery strategy, bolting for home, would be to start from $\mathbf{x}^*$, (the solution from the convex envelope method) and proceed towards a local minimum on F (the true cost function). Unlike the initial state $\mathbf{x}^{(0)}$, which was already a local minimum of F and therefore unaltered by hill-climbing, hill-climbing from $\mathbf{x}^*$ will in general bring about some reduction in cost. However, a less direct but demonstrably effective strategy involves gradually changing the cost function from F^* to F, optimising the function continually as it changes.

A sequence of functions $\{F_i, i=0, \ldots, r\}$ is constructed, such that

$$F_0 \equiv F^* \text{ and } F_r \equiv F. \tag{4.20}$$

and the sequence proceeds gradually from F_0 to F_r. Of course, the first of these is convex, and it is the only convex one in the sequence. Optimisation is first done on F_0 (this is the convex envelope method, as before) and then a succession of local optima are found on each of $F_1 \cdots F_r$. Optimisation on each F_i proceeds from the optimum $\mathbf{x}_{i-1}$, reached on the previous function F_{i-1}. As soon as the condition (similar to (4.15)) is satisfied, that

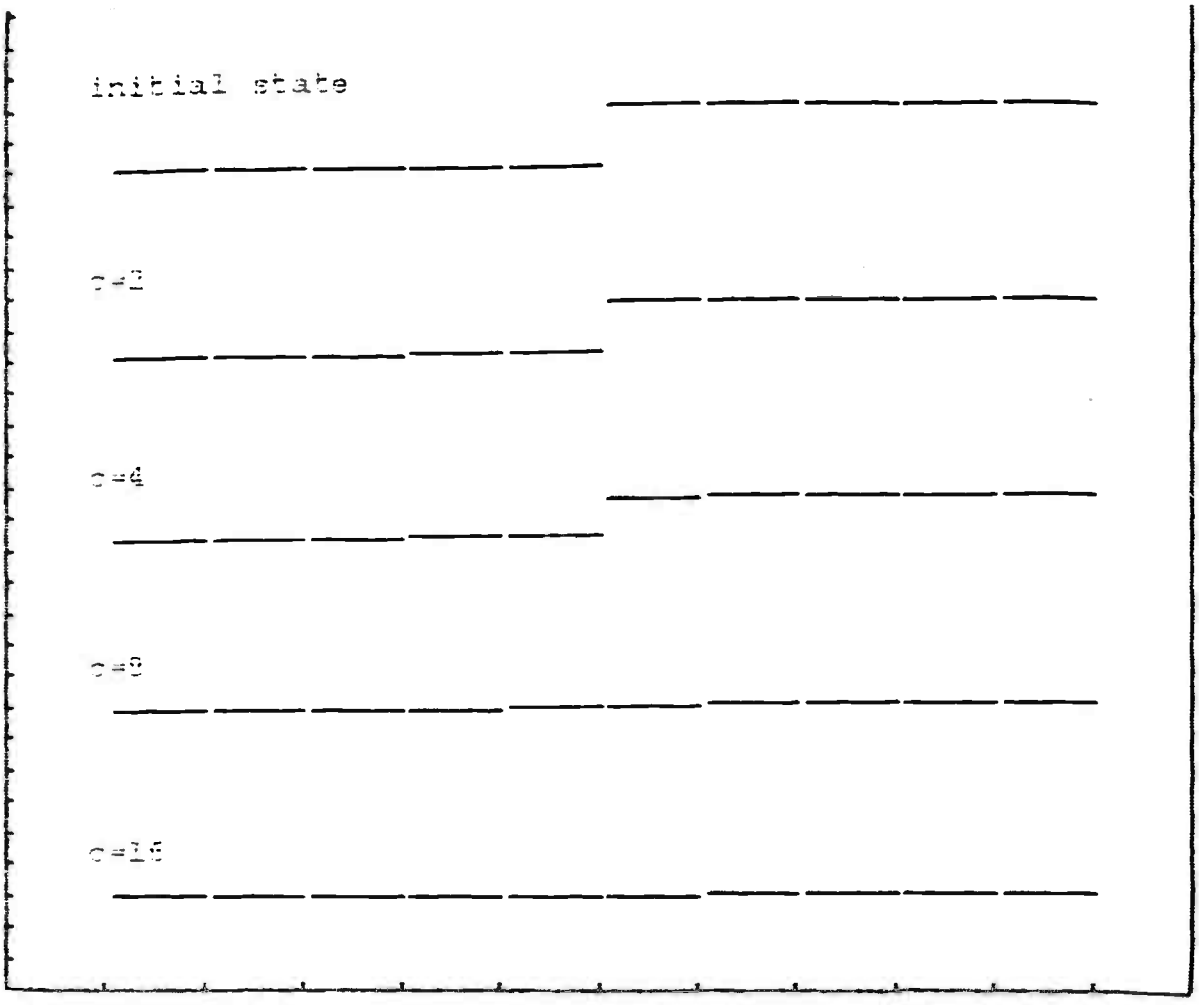


Fig 4.3 The effect of replacing the cost function, F, for weak constancy relaxation in a 10-element linear array, by the convex function F^*. For high or low values of c, the correct result is optimising F^*, but for intermediate values the array settles into a state between the initial labelling and a totally constant labelling.

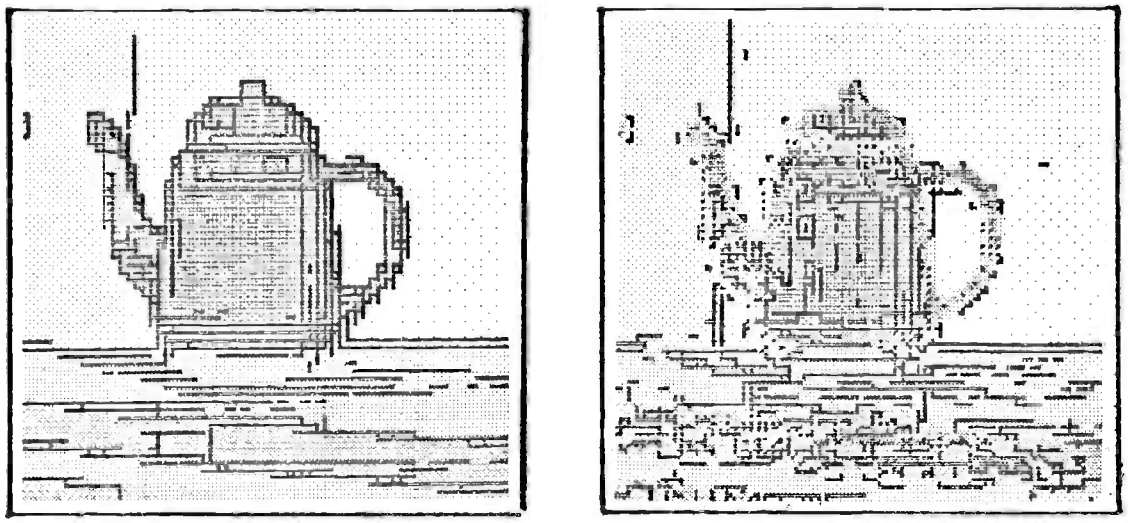


Fig 4.4 Optimisation of the convex function F^*, for weak constancy relaxation on a real image. This procedure finds the correct labelling in some areas of the image but fails in others (see text). Places where it has failed are marked by dark lines. This is shown for two different values, c, of the penalty for breaking a constraint: a) c=128 b) c=32.

$$F_i(\mathbf{x}_i)=F(\mathbf{x}_i), \tag{4.21}$$

a solution has been reached and it can be shown that, for a suitably defined function sequence $\{F_i\}$, the process comes to rest, in the sense that

$$\forall j>i \ \mathbf{x}_j=\mathbf{x}_i. \tag{4.22}$$

The solution reached in this way is not necessarily optimal but may be close to optimal (see appendix D.1). Intuitively, the reason for this is that provided the bulk of the work (i.e. the changes in successive x_i) is done on the first few F_i, which are fairly close to F^*, then this method is "almost the same as" the convex envelope method - and as the convex envelope method (when successful) yields an optimal solution, it is plausible that the solution reached here will be "almost optimal".

We have yet to define exactly how such a sequence of functions can be generated. First a parameterised family of functions, $F^{(p)}, p\varepsilon[0,1]$, is defined, starting at $F^{(0)}=F$ and proceeding smoothly, in some way, towards $F^{(1)}=F^*$. This smooth progression is achieved by the following definition:

$$F^{(p)}(\mathbf{x})=\sum\left(x_{ij}-x_{ij}^{(0)}\right)^2+\sum g^{(p)}\left(\frac{x_{ij}-x_{i,j+1}}{a}\right)+\sum g^{(p)}\left(\frac{x_{ij}-x_{i-1,j}}{a}\right), \tag{4.23}$$

where

$$g^{(p)}=\begin{cases} c\left(p+\dfrac{1}{p}\right)|u|-u^2 \ \text{if} \ |u|<cp \\ c^2 \ \text{otherwise} \end{cases} \tag{4.24}$$

and where, as before, $a=\sqrt{8}$. This definition is chosen to ensure that:

- $g^{(p)}$ is a continuous function
- $g^{(1)}\equiv g^*$
- $g^{(p)}\to g$ as $p\to 0$ (where the function limit here is based on the euclidean function norm)
- $g^{(p)}$ is a steady monotonic progression, in the sense that $\forall p\geq q$, $\forall u$, $g^{(p)}(u)\leq g^{(q)}(u)$.

The function $g^{(p)}$ is sketched for various values of p in fig 4.5. We refer to p as the "non-convexity index". Qualitatively, the neighbour interaction function, $g^{(p)}$, gives a stronger, shorter-range interaction, as the index moves towards 0.

As a matter of practicality, a finite subset of the infinite family $\{F^{(p)}\}$ is selected to form the required sequence of functions The sequence used in the implementation of the algorithm, in this thesis, is

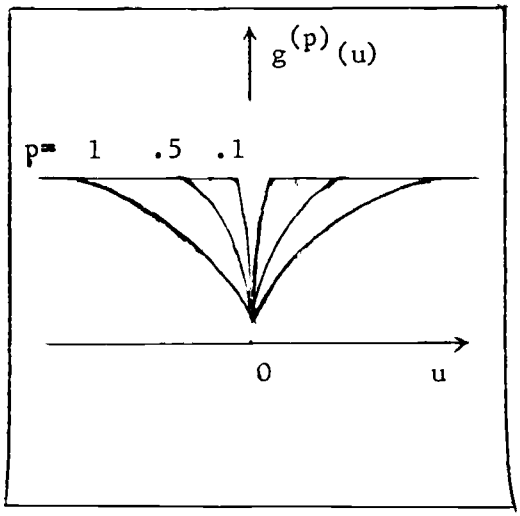


Fig 4.5 The graduated non-convexity method. To achieve a good sub-optimal solution to the minimisation of F, optimisation is performed on each of a sequence of cost functions. These cost functions are constructed using a corresponding sequence of neighbour interaction functions, $g^{(p)}$, which range from $g^{(1)}=g^*$ to $g^{(0)}=g$ (see fig 4.2).

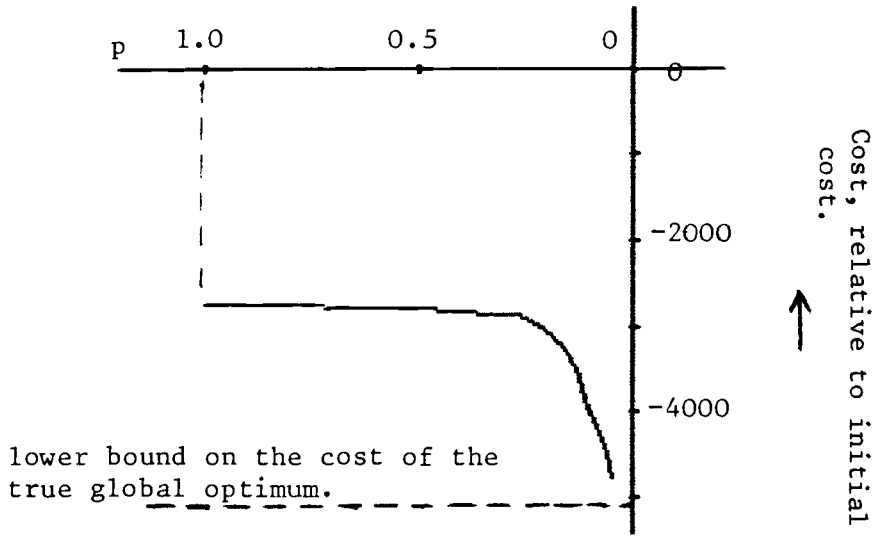


Fig 4.6 The graduated non-convexity method applied to a real image (teapot, with c=64). As the index, p, of $g^{(p)}$, moves towards 0, the cost of the labelling, F(x), decreases.

$$\{F_i, 0 \le i \le r\}, \; s.t. \; F_i = F^{(\lambda^i)}, \tag{4.25}$$

for some constant $\lambda \varepsilon [0,1]$, typically 0.5.

To see how the graduated non-convexity method works in practice, let us consider first a discontinuity in a linear array - the same example for which the convex envelope function was successful sometimes (fig 4.3). It can be shown (appendix D 2) that, for any step discontinuity, the graduated non-convexity method achieves a correct solution - except that the effective value of the penalty constant, c, is a factor $\sqrt{2}$ higher than it should be. In fig 4.6, results are given for the graduated non-convexity method, applied to a real image The graph shows the cost decreasing with successive optimisations, as the non-convexity index decreases A lower bound on the cost of the true global minimum is shown on the graph. (In appendix D.1 it is shown that $F^*(\mathbf{x}^*)$ is such a lower bound.)

It has been shown how the initial non-convex optimisation problem can be solved, sub-optimally, as a particular sequence of optimisations. In turn, those optimisations are to be done by hill-climbing. The optimisations are over a many-dimensional space of variables, and because the functions to be optimised are all sums of local functions, the hill-climbing can be done by relaxation - using parallel local processes (see section 3.3). The next section deals with the details of the hill-climbing algorithm and its convergence properties, and specifies the local processes.

4.2 Relaxation algorithm

Optimisation under weak constraints is achieved, as we saw in the previous section, by hill-climbing on each in turn of a family of functions $\{F_i\}$. How can this be done in parallel, and efficiently? Basically the method is to alter each of the components x_{ij} of $\mathbf{x}$, only taking steps that decrease cost. The x_{ij} are tested, singly or in parallel, and increased or decreased by a certain amount if the cost can be reduced by so doing By computing the actual change in cost that results from an alteration of a component, it is possible to ensure that the cost decreases monotonically. This means that the hill-climbing procedure is bound to come to rest eventually and this avoids the oscillatory behaviour at convergence that is typical of gradient following algorithms (Wilkinson (1963)) Convergence is easy to detect. the process just stops.

Much of this section considers the effect of various parameters on performance and execution time of the algorithm. The data on which observations are based are given in appendix D.5. They are results of a version of the algorithm, written in the C language, that can be thought of as simulating an asynchronous parallel machine (section 4.4), rather than the implementation in the PARAPIC language, emulating a (synchronous) parallel array processor (section 4.3). The reason for this is simply practicality: on a PDP 11/24 the C version runs about 200 times faster. Partly this difference in execution speeds is due to details of implementation such as language efficiency, virtual memory usage etc.. A large part is significant and arises from various losses of efficiency caused by limitations of the parallel array processor Reasons for these losses are dealt with in sections 4.3 and 4.4.

4.2.1 Local computation

From (4.23) it is plain that in order to calculate the change in $F^{(p)}$ due to a change in x_{ij}, it is only necessary to examine 5 local terms: the restoring force due to the energy term $\left(x_{ij}-x_{ij}^{(0)}\right)^2$ and the energies of interaction with each of the four immediate neighbours:

$$g^{(p)}\left[\frac{x_{ij}-x_{i\pm1,j}}{a}\right] \text{ and } g^{(p)}\left[\frac{x_{ij}-x_{i,j\pm1}}{a}\right].$$

In this sense, the evaluation of the change in energy is a local computation, a function only of the element x_{ij} and its four immediate neighbours. In practice, the change in each of these terms can be calculated from the gradients for the functions, plus a small correction - necessary for quadratic functions but not linear ones. For instance, for the quadratic function $f(x)\equiv x^2$:

$$f(x+\delta)=f(x)+2x\delta+\delta^2, \tag{4.26}$$

where the second term on the right is the increase calculated from the gradient and the third term is the small correction. Since the cost function $F^{(p)}$ is made up of a sum of squares (quadratic) and $g^{(p)}$ terms (piecewise quadratic and linear), calculation of local gradients and use of the correction for quadratics as in (4.26) suffices to compute any change in cost due to altering an x_{ij}.

4.2.2 Smoothness of the cost function

There is an important assumption in this method, that if the cost can be reduced by moving from $\mathbf{x}$ to $\mathbf{x}'$ then this can be discovered simply by testing individual components x_{ij}. However, this assumption is by no means valid for all functions, as the example in fig 4.7 shows. In that case the pyramidal shape has a crease in it, which causes trouble: moving either in the x or the y direction from a point on the crease fails to increase f, but moving at a 45 degree angle does cause an increase in f. The crease is a discontinuity in the first derivative of f, and in fact a sufficient condition for the required good behaviour of a function f is that it be continuously differentiable (see appendix D.3).

In the case of weak constancy relaxation, the neighbour interaction function, $g^{(p)}(u)$ has a gradient discontinuity at $u=0$. This can be removed by replacing part of the function, in some neighbourhood $(-d,d)$ of the discontinuity, with a quadratic spline. This is shown in fig 4.8. The new formula for $g^{(p)}$, which was (4.24), is now.

$$g^{(p)}(u)=\begin{cases} \tfrac{1}{2}c\,(p+\frac{1}{p})(d+\frac{u^2}{d})-u^2 \;for\; |u|<d \\ c\,(p+\frac{1}{p})\,|u|-u^2 \;for\; d\le|u|<cp\,. \\ c^2 \;otherwise \end{cases} \tag{4.27}$$

This is a good approximation to $g^{(p)}$ as defined previously, provided p is not too close to 0.

To choose the value of d, arithmetic precision must be taken into account. Integer, rather than floating point arithmetic is desirable to avoid excessive processing and use of storage space. The smallest possible change in any x_{ij} is then a unit change, and it is on this scale that the cost function should appear to be differentable. The constant d should be sufficiently large compared with the unit change. The effect of the constant d is quantified in appendix D.3, where it is shown that the ratio of d to the unit distance can be regarded roughly as a characteristic distance (measured in pixels) over which cells (pixels) can communicate in the relaxation process. There is, however, a limit to how large d can be: the larger it is, the more the neighbour interaction function $g^{(p)}$ becomes distorted. Therefore, d should remain small relative to the total contrast range of the image (i e. the total range of values in the array $\mathbf{x}$). Typically, for images of size 64×64 and 12 bit image data, $d=16$ *units* has been used. Therefore d is $\frac{1}{256^{th}}$ of the maximum contrast range.

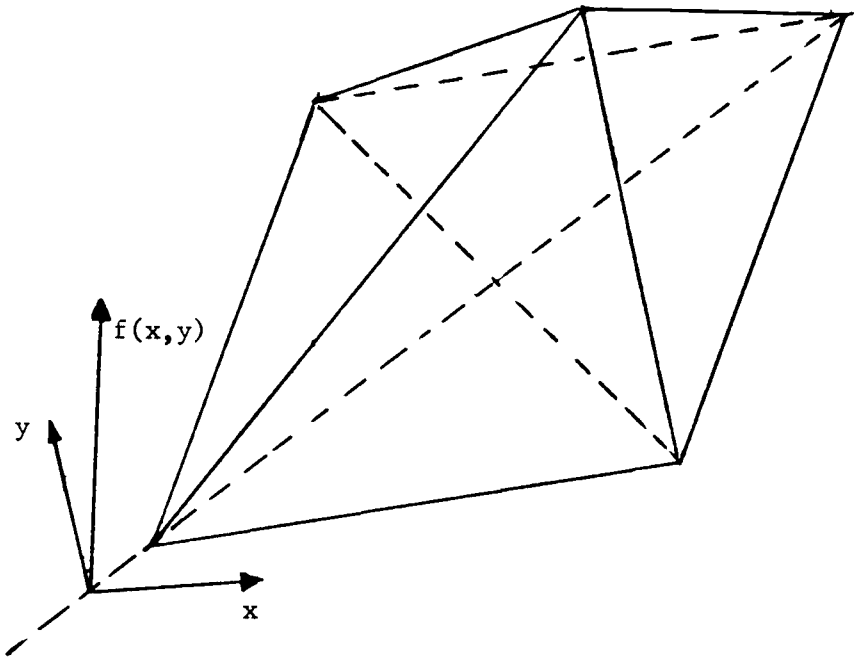


Fig 4.7 An example of a function f(x,y) for which hill-climbing along each of the co-ordinate axes in turn, does not work. From any point on the edge of the pyramid, moving in the x or y directions produces no increase in f. But moving at 45^{o} (along the edge) does increase f.

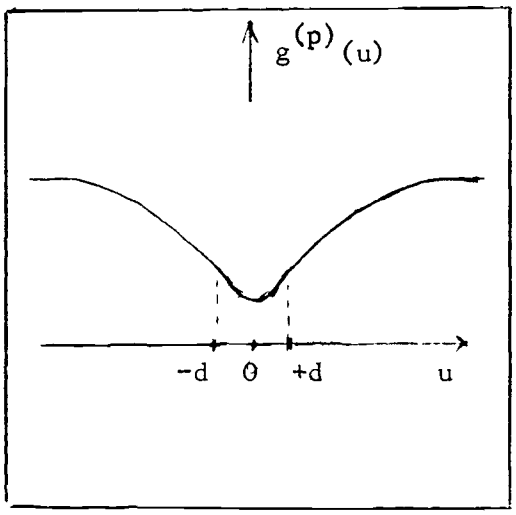


Fig 4.8 To facilitate hill-climbing on a function, the function must be smooth. The neighbour interaction function, $g^{(p)}(u)$ (fig 4.5) is not smooth at u=0. This can be remedied by replacing it, in the region (-d,+d), by a quadratic spline.

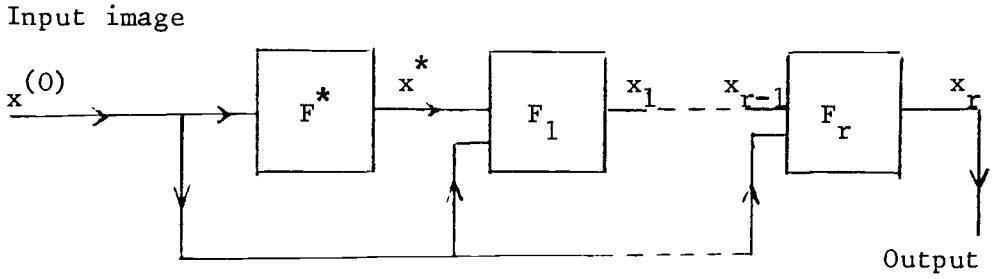


Fig 4.9 A modification to the relaxation algorithm (see text) makes the optimisation of each function in the sequence F_i, i=1..r, convex. In that case, the whole relaxation process can be regarded simply as a cascade of modules in which the output of one module feeds the input of the next.

The value of d does have a small effect on execution time (table 1, appendix D.5). Slightly more execution time is used with higher values of d.

Finally, it is important to remember that the factor, d, must be taken into account when labelling discontinuities, after relaxation has been done. Instead of assigning a discontinuity label between pixels (i,j) and (i,j+1) whenever

$$x_{ij} \neq x_{ij+1}$$

the condition is now

$$|x_{ij} - x_{ij+1}| > d \tag{4.28}$$

- this allows for the alteration to the function $g^{(p)}(u)$ near $u=0$. As mentioned above, provided d is not too large relative to the overall contrast range of the image, varying d should make little practical difference to the labelling.

4.2.3 Precision

Use of high precision carries a heavy penalty in required computation. Results in table 2, appendix D.5, show that, at 12 bit precision, between 4 and 6 times as many iterations are needed as at 9 bit precision. The benefit of high precision is that constancy can be maintained over larger distances: for instance, with d=16, 12 bit precision, variation in a nominally constant region of, say, 10 pixels diameter is less than $\frac{1}{25^{th}}$ of the full contrast range of the image (because in a "constant" region the difference in values of adjacent pixels is not more than d, as in (4.28)); the corresponding variation is as much as $\frac{1}{3^{rd}}$ of the total contrast range - unacceptable in a nominally flat region.

Using adequate precision is essential, but it is not clear why increased precision is so expensive in execution time. This certainly calls for further investigation.

4.2.4 Cost function sequence

The number of functions in the $F^{(p)}$ sequence has a considerable effect on the suboptimality of the final labelling. An upper bound on suboptimality of a particular labelling can be obtained, as described in appendix D.1, by comparing its cost $F(\mathbf{x})$ with the cost $F^*(\mathbf{x}^*)$ ($\mathbf{x}^*$ is the labelling obtained after the initial optimisation on the convex function F^*). The variation in these costs, for varying numbers of $F^{(p)}$ in the sequence, are presented in table 3, appendix D.5. A

larger sequence of $\{F^{(p)}\}$ (with successive $F^{(p)}$ consequently closer together) does indeed result in a more nearly optimal solution, and this is what the theoretical analysis of suboptimality, in appendix D.1, would suggest. The "standard" sequence used for most results in this thesis is

$$\{F^{(1)}, F^{(0.5)}, F^{(0.25)}, F^{(0.125)}, F^{(0.0625)}\}$$

- the convex function and four non-convex ones.

4.2.5 Coarse-to-fine adjustment

A coarse-to-fine progression in size of adjustments made to the x_{ij} (step size) is used in the relaxation process. This proves to be effective in reducing computation time (table 4, appendix D.5). This is because coarse adjustment enables large changes to be made rapidly, initially, and then fine adjustment achieves the precision required in the final labelling. Initially, hill climbing is done with large steps, until no further step can be taken to reduce the cost. Then the step size is halved, there is more hill climbing, and so on, down to unit step size. Step sizes 16:8:4:2:1 are used and there is little to be gained (given the other parameters set as specified in appendix D.5) in using any larger step size.

4.2.6 Penalty for breaking a constraint

The size of the penalty, C ($=c\sqrt{8}$, rescaled for computational convenience) for breaking a constancy constraint has considerable effect on the amount of computation required (table 5, appendix D.5). This is intuitively reasonable, in that large values of C result in greater changes to the initial data - the constraints are stronger. A value $C=128$ results in a 2 or 3-fold increase in computation time and in number of iterations, compared with $C=16$.

4.2.7 Complexity

The variation of computation required with change in the size of the input - that is, with the array size - is shown in table 7, appendix D.5. The number of iterations needed does increase (less than linearly) with linear array size (i.e. N, for an $N\times N$ array). This is plausible because a larger array means larger distances between pixels, and potentially greater distances for communication between pixels. Communications travel at one pixel per iteration and hence more iterations are needed. Execution time for one iteration, in an asynchronous parallel implementation (see section 4.4), might be expected to vary as array area. Overall, therefore, given the increase of number of iterations

required with increasing array size, the overall dependency should be somewhere between the square and the cube of the array's linear size. This situation is complicated by the fact that the execution time for one iteration is strongly dependent on the proportion of pixels in the array that require adjustment (see section 4.4) and in practice, overall execution time seems to be approximately proportional to the array area - the square of the linear size. This observation is based on a somewhat limited quantity of data, and anyway must be somewhat dependent on the image itself.

The fact that execution is so much faster at small array sizes suggests that multilevel relaxation might reduce the required computation. This is a relaxation process that operates over several array sizes; Terzopoulos (1982) reports considerable reduction in execution time, in a surface fitting scheme, by using multilevel rather than single-level relaxation. He obtained a saving in work of a factor of 100, by using 4 levels of array with linear sizes in the ratio 1 2:4:8. Glaser (1983) also discusses multilevel relaxation for solving linear differential equations, in particular solving Laplace's equation for fitting minimum energy surfaces to fixed boundaries (see chapter 1). Again using four levels in the ration 1:2:4:8 of linear array size, a substantially improved rate of convergence is achieved. However, it appears from his theoretical discussion that the operation of the multilevel scheme relies on the linear nature of the equation to be solved. In the case of weak constancy relaxation there is no such linear equation. Moreover, surface fitting involves constructing smooth surfaces, for which sampling errors as a result of moving from a fine to a coarse grid should be fairly small But under weak constancy, surfaces are not smooth; they include discontinuities which would undergo considerable pointwise distortion in sampling. This could be a hazard for a multilevel scheme. However, as potential savings in execution time are substantial, multilevel relaxation under weak constraints is certainly worth investigating.

4.2.8 Ordering dependence

In the synchronous, parallel-array implementation of weak constancy relaxation (section 4.3) array cells are adjusted simultaneously and in parallel. In a single processor or asynchronous parallel implementation (section 4.4) there is some choice about the order in which cells are adjusted. In fact the C language, single-processor implementation uses raster scanning (left to right, top to bottom) but it may be that other scanning schemes could achieve faster convergence. For example, an alternate backward and forward scanning scheme has

been tried with real images, and seems to give slightly reduced execution time.

Note also that with the non-convex function $F^{(p)}$ the local minimum attained depends on the route taken to reach it, so the final solution reached could be dependent on the scanning scheme used However the argument that a good suboptimal solution is achieved (appendix D.1) holds, regardless of scanning scheme.

This dependence of the final labelling on the route taken to reach it can be avoided altogether by a certain modification to the relaxation algorithm that converts the non-convex optimisations of the $\{F_i\}$ sequence into convex ones. Moreover, if this is done, the result $\mathbf{x}_i$ of the optimisation of F_i can simply be fed, together with $\mathbf{x}^{(0)}$, as input to the process of optimising F_{i+1}. The successive optimisations can simply be regarded as a cascade of modules (fig 4 9) - a pipeline that runs continuously. The need for synchrony between modules is avoided; one module need no longer wait for the output of the previous one to be available as a starting point for its own process.

The way in which the optimisation of each F_i is made convex is to use $\mathbf{x}_{i-1}$, not as the starting point for optimisation, but to impose certain constraints on the optimisation process. The function F_i was defined in terms of the interaction function $g^{(p)}(u)$. The function $g^{(p)}(u)$ is composed of three convex pieces which, from (4.24), are: $u<-cp$, $|u|<cp$ *and* $u>cp$. The strength of interaction between any pair of pixels in $\mathbf{x}_{i-1}$ lies on one of those pieces If interaction between those pixels is constrained to remain on the same piece throughout the optimisation of F_i, that optimisation is convex. If, at any point in the optimisation process, this condition is not being obeyed, this is easily remedied by resetting the two pixels concerned to the values of the corresponding pixels in $\mathbf{x}_{i-1}$. This scheme has not actually been tried but, despite the extra constraint that has been imposed on the optimisation of each F_i, the suboptimality argument (appendix D.1) still holds - the method should still achieve a good suboptimal solution

4.2.9 Economy scheme

To cut execution time to the minimum, economy parameters can be used:

d=8 (instead of 16)

10 bit precision (instead of 12)

2 non-convex functions in the sequence, so the sequence is $\{F^{(1)}, F^{(0\,3)}, F^{(0\,09)}\}$.

This economy regime saves a factor of about 4 in number of iterations and in execution time. The results on some real images can be compared in appendix F. The deterioration is clearly visible, particularly for large values of C, but may still be acceptable. These execution times can therefore be taken as minimal. They range from 50-130 seconds, 70-400 iterations, for values of C in the range 16-128.

4.3 Parallel synchronous implementation

This section deals with the parallel, synchronous implementation of weak constancy relaxation. The synchronous architecture considered is the parallel array processor - machines like the CLIP4 (Duff (1978)) and the DAP (Gostick (1979)). The PARAPIC high-level language, fully described in appendix E, has been developed by the author to provide a natural programming environment for parallel array processing, in which the basic operations provided in the language constrain the user to write efficiently parallel-executable programs.

PARAPIC is an extension of the POP-2 language (Burstall et al. (1971)) to include image data types (both boolean and integer) and a number of extra operators and functions. Most of the standard integer and logical operators in POP-2 are overloaded so that they also act on boolean and integer images, to perform their operation in parallel. So, for example, the + operator adds, in parallel, corresponding elements in two integer arrays to produce a third, and the > operator performs a parallel comparison between two integer arrays, producing a boolean array. There are two important new operations in the language, used only with arrays. The first is the parallel conditional operator ?, used in an expression like

$$(B?(I_1,I_2)),$$

which evaluates

$$(IF\ b\ THEN\ i_1\ ELSE\ i_2),$$

in parallel, across the boolean array B and integer arrays I_1 and I_2; the result is an integer array. The second operator is the shift, which has no counterpart in boolean/integer arithmetic, and which shifts a boolean or integer array in a specified direction (North, South, East or West). Special facilities are included for treatment of pixels on the border of the array. Many other operators and

functions are built into the language and described in appendix E.

The PARAPIC language drives an emulator of a parallel array processor, as a separate processor. The emulator itself, written in the C language (1978), is not modelled strictly on one particular array processor but is deliberately restricted to perform functions that can be executed efficiently by the CLIP4 and largely also by the DAP. The CLIP4 is a square array of 96×96 simple boolean processors. Each processor has access to 32 bits of storage and is connected to its eight nearest neighbours. All processors act simultaneously and identically because they are connected to one common instruction bus It is therefore natural to consider all processors en bloc, as one compound processor for 96×96 arrays. Storage can be configured as 32 boolean arrays or, by stacking boolean arrays, they can be treated bit-serially as integer arrays. There is sufficient storage for only 4 8-bit or 2 16-bit arrays and this is inadequate in practice. Such limitations are not to be taken too seriously at the current pace of advancing technology, so the emulator has been constructed with 16 boolean arrays and 16 8-bit integer arrays (any two of which can be traded in for a single 16-bit integer array). Where CLIP4 execution times are given in this thesis they are estimated from the figures published in Duff (1978) and do not include any overheads that might be incurred by a host processor. The time accounted, for example, to add two, 16-bit integer arrays is $160\mu s$.

The weak constancy algorithm has been successfully implemented in PARAPIC. The algorithm takes the form of a sequence of adjustments to an initial image array (using 16-bit precision arithmetic) and the adjustments are carried out in parallel However, care must be taken to avoid undesirable interaction between any cell and its neighbours. Neighbouring cells are defined, in this context, to be those whose values are mutually subject to weak constraints - in the weak constancy algorithm 4-connected pixels are neighbours. To see how interaction can occur, assume that the state of the pixel (i,j) and its neighbours, at the n^{th} iteration, is such that the adjustment $x_{ij}^{(n+1)}=x_{ij}^{(n)}+1$ would result in a reduction in cost at the $n+1^{th}$ iteration. The reduction is calculated on the basis of the values on the neighbour pixels *at the* n^{th} *iteration*. If any of these neighbour cells also have different values at the $n+1^{th}$ iteration the calculated cost reduction may not be realised. One solution to this problem is to partition the pixels into sets, such that no two members of any one set are neighbours. For weak constancy relaxation, where 4-connected pixels are neighbours, just two sets are required:

$$S_1=\{(i,j):i+j \text{ is even}\} \text{ and } S_1=\{(i,j):i+j \text{ is odd}\} \qquad (4.29)$$

They correspond to the black and white squares, respectively, of a chess-board. The iterative algorithm then alternately updates members of S_1, then members of S_2. This avoids interaction and ensures a monotonic reduction of cost in successive iterations Note that the chess-board mask S_1, and its complement, S_2, can very efficiently be generated on CLIP4 using its propagation facility (Duff (1980)).

The border of the array clearly needs special treatment - this was reflected in the form of the cost functions $F^{(p)}$, in which border pixels have only three neighbours or two at corners. This could, in fact, be catered for without incurring any extra computation, by programming CLIP4 at the assembly language level and making use of its hardware border flags. PARAPIC does not give direct access to the flags, a little extra computation is therefore necessary.

The PARAPIC program for weak constancy relaxation is listed in appendix G. Its estimated execution time on CLIP4 is 26 ms per complete iteration - 13 ms to update the black squares of the chess-board and 13 ms for the white squares. From table 6 of appendix D.5 it appears that at least 216 iterations are required at c=64, for 64×64 images. In that case, execution time for the algorithm is 5.6 seconds. This is a least an order of magnitude too slow for real time applications in robotics, where reaction times of a second or less are desirable, and it may anyway be necessary to run the algorithm more than once on each image (at several values of penalty, c, for instance). However that order of magnitude promises to be made up by subsequent array processor designs, such as the GRID (1982), at reasonable cost

Array processor emulation, as used by PARAPIC, proves to be rather expensive. One iteration of weak constancy relaxation takes about 1½ minutes on a PDP 11/24, with 64×64 images, and hence about 5 hours for the complete algorithm. Therefore the results displayed in this thesis were obtained from the very much more efficient (about 200 times) C language implementation, discussed in the next section. The C implementation can be regarded as an emulation of an asynchronous machine. Much of the increase in efficiency reflects quite fundamental inefficiencies in parallel array processing which could be overcome in an asynchronous, parallel machine. The sources of this inefficiency are identified in the next section.

4.4 Parallel asynchronous implementation

Finally, in this section we examine the advantages in using an asynchronous parallel architecture, rather than a synchronous parallel array processor, for relaxation.

The parallel array processor is a Single Instruction Multiple Data stream (SIMD) machine, with one processor to serve each cell in an array. A processor has access only to its own cell and immediate neighbours. The configuration of processor cells is fixed in an array format and the Single Instruction restriction, that there is one common instruction bus, means that all cells perform the same function at the same time. The asynchronous architecture that we consider acts, as before, on arrays of data, but there are relatively few processors and they are each more powerful. Each processor executes a stored program independently of the others although the program is the same for each processor. This could be achieved by having multiple copies of the program available to each processor or alternatively a common program but independent program counters. Each processor has independent, random access to array data so that data cells in need of attention can be served by any free processor. It is assumed that some mechanism is incorporated for calling a processor to serve a waiting data cell. Because there are relatively few processors, it should be practical to make them powerful, having the ability to perform integer arithmetic rapidly (e.g. a 16-bit addition with one operand fetched from memory, in 1 μs). Each processor should also have a plentiful supply of local memory, private to that processor.

Three factors make relaxation, of the type considered in this thesis, execute more efficiently on the asynchronous machine: the multiple instruction stream, random access to array data (especially when the data is sparse) and availability of copious local memory.

The C language implementation of weak constancy relaxation can be regarded as a single-processor, asynchronous machine and its execution times can reasonably be extrapolated to multiple processors. In the previous section, the figure given for CLIP4 to execute 216 iterations of weak constancy relaxation is 5.6 seconds. The corresponding figure for the C implementation is 88 seconds, using a PDP 11/24 processor. With a processor capable of performing a 16-bit addition in 1 μs, this figure becomes 21 seconds. This is based on 64×64 arrays, so to make it comparable with CLIP4 on 96×96 arrays, this time should be multiplied by a factor of the ratio of the areas of the arrays, to give 47 seconds†.

† It was not possible, on the PDP 11/24, to run the C implementation on 96 x 96 arrays because of memory limitations.

Therefore, for our asynchronous machine to give performance, for weak constancy relaxation, equivalent to CLIP4, just 8 or so processors would be needed.

4.4.1 Multiple instruction stream

The importance of the multiple instruction stream - the independence of the processors in executing the program - is that it avoids the inefficiency in executing a 'case' statement that occurs in a parallel array processor. With a parallel array processor, because it is SIMD, *every* cell must go through the motions of executing *every* action, even though only *one* action is applicable at that cell. The effects of all but one of the actions are masked or inhibited in some way, but the cell is nonetheless tied up for the duration of execution of the inapplicable actions, unable to perform useful work. This inefficiency does not arise with the MIMD processors, as each processor executes only the single action appropriate to the cell that it is serving.

Here are two examples of occasions where this saving is significant Some special action is often required at the boundary of an array, in order to impose certain conditions there - to cope with the fact that cells on the boundary have some missing neighbours. In a synchronous machine, all cells inside the boundary are held up while the special border actions are performed. The other example is the evaluation of a function composed of pieces, such as the neighbour interaction function $g^{(p)}$. Each piece of the function is associated with its own arithmetic expression. Initial testing of the value of u establishes which piece applies and the corresponding expression is evaluated In a synchronous machine all the expressions must be evaluated, even though the value of only one of them is required.

4.4.2 Random access, sparsity and interaction

Frequently, in relaxation, the array data is sparse, in the sense that only a small number of the data cells require processing. This can occur for two reasons The first is that the relaxation process itself may be restricted within some subset of the array, as in angle relaxation (section 2.1.3) in which relaxation takes place only along lines in a line drawing. The second is that constraints may, at some stage, already be largely satisfied and only a few alterations remain to be made - this situation arises as relaxation begins to converge.

In the SIMD processor, if just a few cells require attention all other processor cells must mimic those few processors that are actually doing something useful. In the asynchronous machine, only those data cells requiring adjustment

need use any processing resources. The way this is achieved is to maintain a boolean array of 'activity flags' which, if set, indicate that the corresponding data cells need service. In angle relaxation this array remains unaltered throughout relaxation; flags for cells lying on lines in the line drawing are set, and all others are cleared. Processors only serve cells whose flags are set. For relaxation in a full array - an intensity array, for example - the same activity flag array is used, but flags are altered in the course of relaxation. Initially, all flags are set because any cell may require adjustment. When a data cell is processed and found to need no adjustment its flag is cleared, but when a cell receives an adjustment, not only does its flag remain set but the flags of all its neighbours are set too. This is because changing the state of a cell can also change the state of its neighbours, due to the interaction between them. Use of activity flags in weak constancy relaxation has been found to save, overall, a factor of about 4 in execution time, for real images.

Related to this is the problem of *interaction*: two neighbouring cells, both in need of alteration, must not be processed simultaneously - one must wait for the other. This problem was discussed with reference to parallel array processors in section 4.3; there the proposed solution was to split the array into two non-interacting sets - the black and the white squares of a chess-board. Alterations may only occur in one set at any one time. For the asynchronous machine, a natural solution would be to extend the use of the activity flag array to perform a semaphore function in which any cell being processed locks out its neighbours until processing is complete.

4.4.3 Local memory

The availability to each processor of a large amount of local memory makes an important contribution to efficiency; the repeated evaluation of complex expressions can be avoided by using look-up tables. In the C language implementation of weak constancy relaxation, an 8K byte table is used to compute the neighbourhood interaction function and this results in a considerable saving in execution time.

5 Parallel algorithms for least disturbance labelling under linear constraints.

The purpose of chapter 2 was to explain how certain tasks in low level vision can be specified in terms of constraint satisfaction. In particular, intrinsic images (Barrow and Tenenbaum (1978)) are a natural framework in which constraints can be applied. Each intrinsic image represents a quantity such as intensity, reflectance, surface orientation, and constraints apply both between images (inter-image) and within images (intra-image). The main contribution of this thesis is in dealing with intra-image continuity constraints and this has been fully expounded in chapters 2 and 4.

Some inter-image constraints can be expressed in a linear form and algorithms are needed to impose them. This chapter discusses the design of such algorithms, using the least disturbance principle - that a minimal adjustment should be applied to initial data, in order to satisfy the constraints. In section 2.3.3 we saw that this can be expressed as a constrained optimisation problem, and the solution, we now show, can be achieved by an iterative, parallel, local, linear algorithm. The algorithm here is related to the previous work by Ullman(1979), Horn and Schunk (1980) and Burch et al. (1983), that was outlined in section 3.3.

In the course of chapter 2, the edge-region duality constraint was described. It proved to be inadequate as an expression of intra-image continuity - weak continuity constraints were much more effective. But edge-region duality, expressed by linear constraints, is a useful test-bed for the design principles for linear parallel algorithms developed in this chapter. An algorithm for imposing edge-region duality is developed and implemented using the PARAPIC language (appendix E.) and executed on an emulator of a parallel array processor. Simulated execution time for a CLIP4 parallel array processor is less than ¼ of a second. The convergence of the algorithm under these constraints is proved. The rate of convergence is discussed theoretically and sheds some light on actual convergence.

5.1 Satisfying linear constraints

We require to solve the optimisation problem of minimising the change in some data (labels on the nodes of a graph), subject to linear constraints. First some notation is needed. The labels themselves are $\{X_{i\lambda}, 1 \le i \le n, \lambda \varepsilon \Lambda\}$ - the value of label λ on the ith node. In the context of image processing, i indexes the pixels of an image. Usually the pixel coordinate would be represented by a pair (i,j), its cartesian coordinates. Without loss of generality but with a considerable gain in clarity, a single index i is used for the time being. We will also refer to $X_{i\lambda}$, dropping the first index, as X_λ. At the ith node (pixel) there are several quantities to be labelled, indexed by $\lambda \varepsilon \Lambda$. The set Λ could define the different quantities represented by intrinsic images, for example we might have:

$$\Lambda = \{intensity,\ surface\ orientation,\ range,\ reflectance\}.$$

Each of these four quantities is represented at each pixel. The array

$$I_i = X_{i,intensity}$$

is precisely the intensity intrinsic image.

A set of n linear constraints (one constraint is attached to each node and acts in the node's neighbourhood) is specified by a set of square (nxn) matrices $C_\lambda, \lambda \varepsilon \Lambda$, whose elements are $(C_\lambda)_{ij}, 1 \le i,j \le n$. The constraints are:

$$\forall i \ S_i = 0, \ where \ S = \sum_{\lambda \varepsilon \Lambda} C_\lambda . X_\lambda . \tag{5.1}$$

$$[Note: \ C_\lambda . X_\lambda \ denotes \ \sum_j (C_\lambda)_{ij} X_{j\lambda} .]$$

Given an initial labelling $X_\lambda^{(0)}$ we define a distance measure T:

$$T = \sum_\lambda ||X_\lambda - X_\lambda^{(0)}||^2 w_\lambda , \tag{5.2}$$

using the euclidean norm, which is defined by

$$||A||^2 \equiv \sum_i A_i^2 . \tag{5.3}$$

The w_λ are scale factors for the different quantities in Λ. This makes the X_λ (intrinsic images) comparable with one another for the purpose of the distance measure T. It would be appropriate to take account, in the normalisation,of the expected deviation from the initial labelling of the quantity λ. Each w_λ can then be regarded as a measure of "stiffness", or reluctance to change, of the quantity λ.

The constrained labelling problem is to minimise T subject to the constraint that $S=0$. The objective function is convex and the constraints are linear, so the global minimum can be found by finding the unique extremum (local minimum). This can be achieved by the method of Lagrange multipliers (see appendix C.1) to give a set of equations that can be solved iteratively, by Jacobi relaxation (appendix C.1). The iterative formula is:

$$\forall\lambda\ X_\lambda^{(n+1)}=X_\lambda^{(n)}-\frac{1}{w_\lambda}C_\lambda^T.D^{-1}.S^{(n)} \tag{5.4}$$

where $X_\lambda^{(0)}$ is the initial data, as before,

$$S^{(n)}=\sum_\lambda C_\lambda.X_\lambda^{(n)} \tag{5.5}$$

and D is the diagonal part of the matrix P, given by:

$$P=\sum_\lambda \frac{1}{w_\lambda}C_\lambda C_\lambda^T. \tag{5.6}$$

(The ^{T} denotes matrix transpose.) All this is justified in appendix C, and the scheme is shown to be convergent if the matrix P satisfies a certain condition (namely, diagonal dominance). This condition is a restriction on the constraint matrices C_λ, limiting, in a certain sense, the degree of overlap or correlation of the neighbourhoods (in X-space) over which each of the constraints $S_i=0$ acts. In section 4.2.2, where this method is illustrated for the edge region duality constraint, the diagonal dominance condition does not quite hold It is therefore necessary slightly to modify the iterative scheme in (5.4) in order to ensure convergence (see section 5.2).

How is this algorithm related to previous work? Ullman (1979) (see also section 3.3) describes a similar scheme that is more general; it caters for linear constraints and also convex non-linear constraints, and with any convex cost function. However his relaxation algorithm is more complex, as it involves iterative adjustment both of label values and of Lagrange multipliers, rather than just the former. Moreover Ullman's algorithm is for inequality constraints, not for equality constraints as here. However a simple specialisation of his algorithm would allow equality constraints to be handled by expressing the constraints

$$S_i=0 \text{ as } (S_i)^2\leq 0. \tag{5.7}$$

These are incorporated into the Lagrangian:

$$L=T+\sum_i M_i(S_i)^2 \tag{5.8}$$

where M_i are Lagrange multipliers. L is convex as required. Ullman's algorithm finds the saddle point of L - that is a minimum in the $X_{i\lambda}$ and a maximum in the M_i. This can be done by an iterative hill climbing process in X and M.

However, in the scheme described in this chapter the special properties of the problem - a simple quadratic cost function and linear equality constraints - result in a relatively simple linear form for the iterative algorithm (5.4). In fact the problem can be viewed in the vector space of $X_{i\lambda}$: to find the minimum distance from a given point (the initial data) to a particular subspace (the space of consistent labellings). The projection theorem (Luenberger (1969)) says that this can be achieved simply by projecting the point onto the subspace. It is relatively straightforward to show that the iterative formula (5.4) achieves precisely this projection (see appendix C.2).

The Horn and Schunk (1980) shape from shading algorithm, also discussed in section 3.3, is similar to the one described here, but relates to unconstrained optimisation. Their distance function is the same as the one used here (Euclidean norm) but in place of constraints they have a smoothness measure which is added to the distance function, to make up the objective function. Moreover they suggest using Gauss-Seidel relaxation (Varga 1962), a serial method; here the method used here is Jacobi relaxation - very similar but executed in parallel over the array, and therefore suitable for a parallel array processor.

5.2 Edge-region duality

In section 2.3 the problem of finding edges in an image was expressed, somewhat inadequately as it transpired, in terms of edge-region duality and minimum strength constraints. We now illustrate how the resulting optimisation problem, under these linear constraints, can be solved along the lines set out in the previous section.

Given an image I we take, as the natural estimate of edge strengths, those strengths $(H^{(0)}, V^{(0)})$ that represent I (see chapter 2.3), obtained simply by differentiating I. This means that initially, edge strengths are simply the differences between adjacent pixels. They are modified according to the principle of least disturbance, to conform to the two constraints defined earlier. This generates an optimisation problem, to find the edge strengths (H, V) which minimise

$$T = \|H - H^{(0)}\|^2 + \|V - V^{(0)}\|^2 \qquad (5.9)$$

subject to the constraints of continuity and minimum strength. The edge-region duality constraint is that $S=0$, where

$$S_{ij} = H_{ij} - V_{ij} - H_{i,j+1} + V_{i-1,j} \tag{5.10}$$

and the minimum strength constraint is that wherever $H_{ij}^{(0)}$ is *inactive* (below some minimum strength in the initial image) then at (i,j) in the final labelling

$$H_{ij}^{(n)}=0, \textit{ and similarly for } V. \tag{5.11}$$

Elsewhere, for *active* edges, H_{ij} is determined by an iterative formula, corresponding to the one in the previous section (5.4). In appendix C.3 we see that this is:

$$H_{ij}^{(n+1)}=H_{ij}^{(n)}-(1-k)\left[\frac{S_{ij}^{(n)}}{r_{ij}}-\frac{S_{i,j-1}^{(n)}}{r_{i,j-1}}\right], \tag{5.12}$$

where $S^{(n)}$:

$$S_{ij}^{(n)} = H_{ij}^{(n)} - V_{ij}^{(n)} - H_{i,j+1}^{(n)} + V_{i-1,j}^{(n)} \tag{5.13}$$

is a measure of the degree to which the continuity constraints are violated. The term r_{ij} is the number of active edges contributing to S_{ij}, in (5.13), so that $1 \le r_{ij} \le 4$. Because the diagonal dominance condition on P, discussed in the previous section, does not hold here, it is necessary to introduce the constant k, $0<k<1$ which should be close to 0 for rapid convergence. Setting $k=0$ in (5.12) is equivalent to the relaxation formula (5.4) derived in the previous section. Inactive edges remain at 0 for all n. The final labelling is the limit:

$$(H,V)=\lim_{n\to\infty}(H^{(n)},V^{(n)}). \tag{5.14}$$

It has been proved that the formula (5.12) converges for $0<k<1$. The derivation of the formula and the proof of convergence extend easily to deal with two or more arrays of labels, under linear homogeneous constraints of a certain type. The derivation and proof are given in appendix C.3.

5.3 Parallel processor implementation

The next section briefly describes results obtained from executing the edge relaxation algorithm on a parallel array processor emulator. Parallel array processors like CLIP4 (Duff (1978)) pose a problem, because of limited storage space for arrays. Floating point or high precision integer arrays may be

unacceptably extravagant; however, care must be taken to retain the convergence of the algorithm and, as far as possible given limited precision arithmetic, the correctness of the solution.

The usual termination condition for iteration in an array is that some measure of the change in the array, in one iteration, falls below a certain value. Oscillatory behaviour can be eliminated provided a bound on its magnitude is known. With arrays of limited precision (8 or 16 bit) it is unlikely that such an upper bound will be sufficiently small to be useful. An alternative strategy in this case, is to partition the labels $X_{i\lambda}$ into sets, so that each S_{ij} in (5.13) depends on exactly one edge from each set. The iteration scheme is also partitioned to adjust labels in one of the sets, recompute S, adjust labels in another of the sets, and so on. In this way each S_i is affected by only one label in any single iteration. This method is a sort of hybrid of parallel Jacobi and serial Gauss-Seidel relaxation: it is parallel within the label sets, but the sets are processed in sequence. It can be shown that, choosing $k > \frac{1}{2}$, convergence is monotonic in the sense that $\sum_i |S_i^{(n)}|^2$ decreases monotonically. Hence non-oscillatory convergence is guaranteed but at the expense of speed; this is because we must choose $k > \frac{1}{2}$ but $k = 0$ gives the fastest convergence In the case of edge relaxation the partitioning of the labels is into four sets (see fig 5.1).

Figure 2.8 shows a picture of a spanner (fig 2.8a), taken under diffuse lighting with a CID camera, and the effect of the edge relaxation algorithm applied to it. The minimum strength constraint requires that all edges, in the initial edge strength labelling, below some low threshold be removed (fig 2.8b); the threshold was chosen at a turning point in the edge strength histogram, and removes the bulk of low strength edges. The application of the continuity constraint, by relaxation, results in a clean edge strength map (fig 2 8c). The effect of the relaxation s both to erode any boundaries in fig 2.8b with unattached ends, and to redistribute edge strengths within the remaining boundaries. The partitioned algorithm for limited integer precision (described above) was used. It was executed on an emulator of a parallel array processor which is programmed in the PARAPIC parallel image processing language (appendix E). The edges in fig 2.8c have a dual representation as regions (fig 2.8d), obtained from the edge strengths by a process of integration. The grey level within each region is constant, as expected, except for some striping which is due to the effect of rounding error in relaxation, accumulated and smeared by the integration.

Convergence of the relaxation process can be traced by plotting the maximum deviation, in the array, from the continuity constraint, at successive

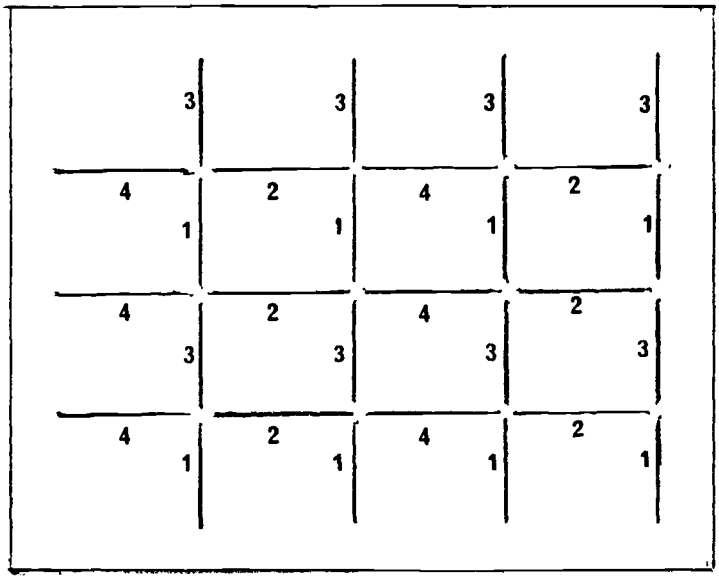


Fig 5.1 A regular array of edges can be partitioned into four non-interacting subsets. Here each edge is labelled 1,2,3 or 4, according to the set to which it belongs. No two adjacent edges belong to the same set. If relaxation adjusts edges only in one set at a time, undesirable interaction between neighbours can be avoided.

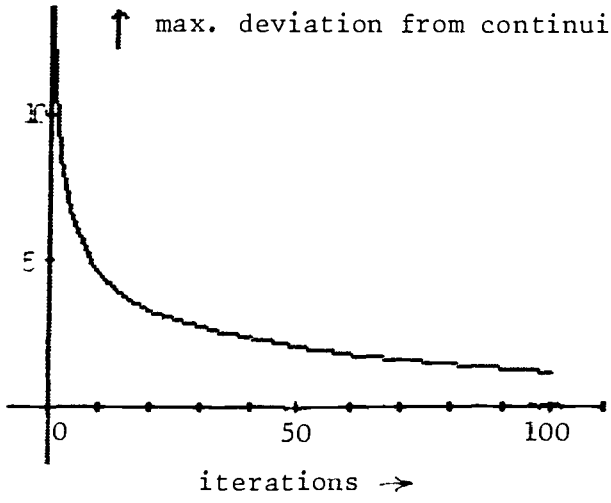


Fig 5.2 Convergence of edge-region duality relaxation. This graph shows the maximum deviation, over the array, from consistency, at successive iterations. The graph is monotonic decreasing, as theoretically predicted. It converges rapidly initially but has a slow tail.

iterations (fig 5.2). A theoretical prediction of what to expect in the case of a closed boundary of M edges, can be obtained (appendix C.4). The rate of convergence is strongly dependent on the initial pattern of edge strengths. Some patterns converge within a few iterations, while in the worst case, the deviation from continuity decays to $\frac{1}{e}$ only after about $\frac{M^2}{\pi^2}$ iterations. The convergence graph (fig 5.2) appears to reflect this, converging rapidly initially, followed by a slow decay. Note also that in the partitioned scheme for limited integer precision, described earlier, the graph is guaranteed to be monotonic decreasing.

How fast can the edge relaxation algorithm be expected to run? Using the figures published for the CLIP4 design (Duff (1978)) but assuming that 144 rather than 32 boolean planes of storage are available, the parallel processor time needed for a 96x96 pixel image is

2.9 ms + 4.1 ms per iteration

- about 410 ms for 100 iterations (as used in fig 5.2).

5.4 Asynchronous linear relaxation

The linear algorithm of section 5.1 is synchronous, in the sense that it is an iterative series of operations, in which each operation acts simultaneously and in parallel at every label. It is essential to perform the label adjustments evenly and at the same rate, rather than, say, perform many adjustments to half the labels and then adjust the other half. If the adjustment is done unevenly a consistent labelling will still be obtained (the constraints will be satisfied) but it is not the least disturbance labelling. As an example, in the edge labelling case, the constraints could be satisfied simply by adjusting horizontal edges, until convergence were reached, without altering the vertical edges at all. The result is consistent but unlikely to be the least disturbance labelling.

The synchronous algorithm is suitable for a parallel array processor, which is a synchronous machine. However, for an asynchronous parallel machine (see section 4.4 and chapter 6) it may be desirable to use an asynchronous algorithm - one that reaches the same final solution, whatever the order in which the labels are updated The initial optimisation in the weak continuity algorithm of chapter 4 - the optimisation on a convex cost function - is asynchronous in this way.

An asynchronous algorithm for satisfying linear constraints can be set up if the linear constraints (5.1) can be expressed in a parametric form:

$$\forall \lambda \; X_\lambda = B_\lambda . Y. \tag{5.15}$$

(B_λ is an $n \times n$ matrix and Y is a vector of parameters.) In this case the constrained optimisation problem of section 5.1 becomes an unconstrained one, to minimise

$$T = \sum_\lambda ||B_\lambda . Y - X_\lambda^{(0)}||^2 \; w.r.t \; Y. \tag{5.16}$$

This is done by hill climbing in Y-space and the order in which the Y_i are altered does not affect the final solution - the position of the summit is quite unaffected by the route followed to reach it. The previous, constrained, linear algorithm started with an initial labelling and moved, by the most direct route, to a consistent one. This unconstrained, asynchronous algorithm starts with a consistent labelling and, remaining consistent throughout, moves as close as possible to the initial labelling.

As an example consider the quantities range(D) and surface orientation(N), related by the constraint

$$N = \nabla D, \tag{5.17}$$

which holds for small variations of D about some constant value. Surface orientation is the spatial derivative (a vector field) of range. In discrete terms, we have D_{ij} and $N_{ij} = (N_{ij}^x, N_{ij}^y)$. The constraint (5.17) becomes:

$$\forall i j \; N_{i,j}^x = D_{i,j} - D_{i-1,j} \; and \; N_{i,j}^y = D_{i,j} - D_{i,j+1} \tag{5.18}$$

- just the same, in fact, as the edge region duality constraint, with D and N instead of I and (H, V). The constraints (5.18) are already in the required parametric form, with D_{ij} as parameters. Hence they can be imposed by an asynchronous algorithm. (Note that although edge-region duality could have been put in parametric form, as above, this is no longer feasible when the minimum strength constraint is also applied.)

If constraints cannot conveniently be expressed parametrically, there remains the option of using the variant of Ullman's Lagrangian scheme, described in section 5.1. By introducing Lagrange multipliers, as we saw, a joint hill climbing process, involving labels and multipliers, can be used. In this way, at the cost of some extra complexity in the relaxation formula, a least disturbance labelling can be achieved, under any linear constraints, by an

asynchronous algorithm.

6 Discussion

This thesis is principally about weak continuity constraints and their application to low-level vision. For example, at an early stage of interpretation, image intensity data may be described in terms of lines. These are lines of discontinuity in the data - lines along which some continuity constraint is broken. As another example, vertices on curves in a line-drawing can also be characterised as discontinuities, in this case discontinuities in the angle of the tangent to the curve. More generally it is useful to locate discontinuities in various quantities (represented by intrinsic images) such as intensity, reflectance, surface orientation and others.

The definition of continuity in a discrete array is discussed in chapter 2; in this thesis a somewhat restricted definition has been used (constancy) in which continuous surfaces are defined to be uniform. However, further work is needed to develop this definition to include regions of extended gradient and other smooth profiles. The knowledge that intensity varies smoothly almost everywhere (except on lines of discontinuity) has been expressed as a weak continuity constraint - a constraint that may be broken on payment of some fixed penalty c^2. These penalties are used to achieve a fit to initial data $\mathbf{x}^{(0)}$, producing a new array $\mathbf{x}$ which minimises the following cost function:

the difference $||\mathbf{x}-\mathbf{x}^{(0)}||^2$

PLUS

sum of penalties for broken continuity constraints.

It has been shown that the minimisation of the cost function just described cannot be tackled directly by hill-climbing. This is because the function is non-convex - it has many local minima. Replacing the cost function by a certain sequence of functions and hill-climbing on each in turn (the method of graduated non-convexity) proves to be a way of reaching a solution that is close to optimal.The resulting algorithm is highly parallel. It can be executed rapidly on a parallel array processor but it seems to be more efficient to use an asynchronous parallel machine. The algorithm has been applied to some real images to label lines and vertices with promising results but more extensive testing is required to establish its robustness.

Two speculative issues that call for further work remain to be discussed. First, what is the scope and what are the limitations of the weak continuity

model in intrinsic images? Secondly what is a suitable architecture for a parallel machine to execute the relaxation algorithms? Results presented in chapter 4 indicated that an asynchronous ensemble of relatively few, powerful, arithmetic processors would be more effective here than a rigidly structured array of synchronous processors. Some of the requirements of such an asynchronous machine will be considered.

6.1 The scope of weak continuity in intrinsic images

The intrinsic image stack requires processes which maintain continuity within each intrinsic image and consistency between images. We have seen how weak continuity can be achieved (chapter 4) and how linear inter-image constraints can be maintained, by relaxation (chapter 5). What is required is to do both together, over several intrinsic image arrays. Provided that the linear constraints can be expressed in a parametric form (section 5.4) they are automatically satisfied simply by using the parameters as variables in hill-climbing. As an example, inferring shape from shading involves the intrinsic images for intensity (I) and surface orientation ($\vec{N}$, a unit vector). If illumination is uniform and consists of l_0 (diffuse) plus l_1 (directional, in a uniform direction $\vec{s}$) then for an unshadowed surface with constant reflectance r:

$$I = r((\vec{s}.\vec{N})l_1+l_0). \tag{6.1}$$

Assuming that the only sensor used is a camera, supplying intensity data I_0. The relaxation scheme could use N_1,N_2, where $N_1=\vec{N}.\vec{s}$ and N_2 is a component of $\vec{N}$ in some fixed direction perpendicular to $\vec{s}$ (they are subject to the convex constraint that $N_1^2+N_2^2\leq 1$) as parameters for hill climbing. Use of these parameters ensures that the optimisation problem is convex.

The cost function would consist of a term $||I-I^{(0)}||^2$, the distance-cost for deviation from the initial data, plus the terms representing weak continuity of $\vec{N}$. It is not necessary or desirable also to apply weak continuity constraints to I as discontinuities in I can only arise from discontinuities in $\vec{N}$ (in this simple scheme). Such a shape from shading algorithm would have the ability to decide the positions of discontinuities in surface orientation because of its use of weak continuity constraints. This goes beyond the capability of the Ikeuchi and Horn (1980) algorithm that makes surface orientation vary smoothly everywhere.

In a more general scheme the constants r and l_1 would be replaced by intrinsic images R and L so that the linear relation (6.1) is replaced by:

$$I=R((\vec{s}.\vec{N})\times L+l_0). \tag{6.2}$$

This is no longer linear and presents a problem because I is a non-convex function of R, L and $\vec{N}$. Some method is needed for dealing with the non-convexity. Perhaps it is necessary to make heuristic restrictions that allow only two variables, I and one other, to vary at any one array cell. It seems that, in practice, the ideal of one single, asynchronous, unsupervised, optimisation process throughout the intrinsic image stack may be overambitious.

Stereopsis is an important low-level vision task and we now consider whether cooperative processing in the intrinsic image framework might have any application there. Stereopsis has been considered as a candidate for cooperative computation (Julesz (1971), Marr (1976)) but more recently arguments have been put forward against cooperative stereopsis (Marr (1982)). Central to stereopsis is the correspondence process which matches features in the left image to features in the right image. The features are discrete and may carry some real-valued measures; matching is essentially some sort of discrete optimisation process to pair features with similar measures, subject to ordering constraints (e.g. Baker (1981)). Once matching is complete it is necessary to excise false matches ("ghosts") - and there may be many of them (Mayhew and Frisby (1981)) - by imposing continuity constraints on binocular disparity. This can be done cooperatively (Baker (1981)) but by a discrete algorithm, rather than by real valued relaxation.

Real-valued relaxation could perhaps assist stereopsis by constructing higher level primitives for matching. For instance, weak continuity relaxation of intensity and then of angle provides, as we saw in chapter 2, primitives in the form of lines labelled with endpoints and orientation. Matching these larger scale primitives should substantially reduce false matches in the subsequent correspondence process. Continuity of binocular disparity, along lines of intensity discontinuity, is guaranteed by the use of these line primitives for matching. Marr (1982 pp 130) suggests that orientation information should be used in matching but Mayhew and Frisby (1981) argue that the human vision system uses unoriented primitives. Their argument is based on the degradation of human stereopsis when an oriented stimulus is masked by oriented noise, at a different orientation. This argument makes an assumption about the effect of superimposed stimuli - that the response of an oriented detector is unaffected

by noise at a different orientation; while this is valid for linear detectors it is not necessarily true for non-linear operations like weak continuity relaxation.

A possible use of relaxation in stereopsis is for interpolation between sparse disparity tokens (rather like spot heights on a geological survey) as described by Terzopoulos (1982). Whether weak continuity relaxation is of any use here depends on the nature of the tokens used in the correspondence process. If they already correspond reliably to intensity discontinuities then all that is required is interpolation between discontinuities. Otherwise, weak continuity relaxation could be applied to an array of disparity values, perhaps jointly with intensity data (using the relation (6.1) and a relationship between range and surface orientation), to mark lines of discontinuity in the disparity image.

Another aspect of intrinsic image processing, mentioned in chapter 2, is the possibility that high level processes might influence a computation performed in the intrinsic image stack. No specific mechanism was proposed for this and two possibilities are now discussed. The first is for a high-level process to control the penalty c^2 to achieve the required level of detail in the labelled discontinuities. There is no particular reason why c must be uniform throughout the image; for instance, if current hypotheses indicate that a certain area is more brightly lit than the rest of the image then it would be appropriate to use a high value of c in that area. The second way that relaxation can be influenced is to constrain a certain area of a certain intrinsic image when a priori knowledge or current hypotheses require it. This could take the form of instantiation of intrinsic image elements in that area, so that they are fixed during the relaxation process, or to impose inequality constraints on them. Use of inequality constraints (especially on range and surface orientation and also on reflectance) is consistent with the way in which the model based inference system ACRONYM (Brooks (1979)) represents generic models and partial information. Conjunctions of convex inequality constraints (including all linear ones) can be imposed in relaxation simply by limiting the domain of hill-climbing - fencing off any area in which some constraint is broken and declaring it out of bounds.

Some important tasks that are not so far catered for by weak continuity relaxation include dealing with surface markings, texture and virtual lines (lines that are inferred as, for instance, in fig 6.1). Weak continuity relaxation characterises edges as discontinuities between continuous regions but this is inappropriate for surface markings which are events in intensity-gradient space, rather than in intensity space itself - that is, "edge events" not "region events".

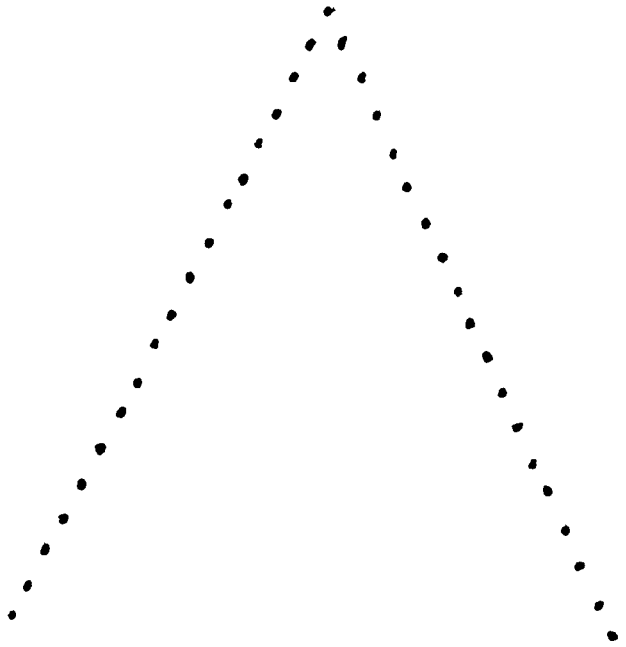


Fig 6.1 A candidate task for weak continuity relaxation. A virtual line is perceived joining up the dots in this figure. Weak continuity, used on the output of a suitable dot-joining operator could be used to define angle discontinuities (vertices) in this virtual line.

Where surface reflectance is modulated by texture, weak continuity relaxation treats it as noise and suppresses it provided its amplitude is not so great as to mask intensity discontinuities. It should be able to discriminate between textures with differing average intensity, but not between those that differ only in their higher order moments. For higher order differences it would be necessary to use texture measures that respond to higher order properties, and then apply relaxation. Finally, virtual lines, as in the Kanisza's triangle illusion (fig 2.1) or as implied in a chain of dots, have not been dealt with. Zucker (1983) proposes a relaxation/optimisation based scheme to deal with dot patterns, in which a consistent angle labelling of a curve is defined to be a local optimum of a certain objective function. In the pattern in fig 6.1 a discontinuity is plainly perceived at the top of the figure; weak continuity constraints on local angle would be appropriate there to label the orientation discontinuity, given some suitable dot-joining operator. Using weak continuity constraints brings the advantages that have been discussed in earlier chapters: the labelling is defined uniquely as a global optimum and the semantics of "continuity almost everywhere" are clearly represented.

6.2 Architectures for asynchronous relaxation

In chapter 4 a somewhat idealised asynchronous architecture was discussed for the purpose of assessing the efficiency of an asynchronous parallel implementation of weak constancy relaxation. The features of this architecture were:

- a number of powerful (one 16-bit addition in a μs) arithmetic processors, but relatively few of them, far fewer than the number of cells in an array.
- each processor has its own copy of a common program and executes it independently of the other processors
- each processor has random access to global data
- each processor has plenty of local memory, enough to use look-up tables for often-used functions

The global memory must be available to all processors. This could be achieved simply by a single bus running from memory, around all processors and with appropriate arbitration logic but this is likely to be slow. In fact for weak constancy relaxation the degradation in performance would not be very severe with up to 20 or so processors on one bus (assuming 200ns access time), because the bulk of computation involved only local memory (which is served by busses

private to each processor). If better access to global memory is required, there are various alternatives: one is to use a parallel switching network between n processors and n global memory modules (Gottlieb et al (1983)) but the network is complex, consisting of $n\log_2 n$ switching units. Another possibility is the ZMOB architecture (Kushner et al. (1982)) in which there is no global memory, only local memory and interprocessor communication via a high-speed conveyor-belt bus. In a relaxation algorithm arrays must be partitioned into areas, one per processor. Relaxation then consists of alternating periods of computation and interprocessor communication. Preliminary estimates indicate that the communications overhead would be acceptably low for weak constancy relaxation. However the ZMOB conveyor-belt bus is also fairly complex (the prototype ZMOB hardware cost is 150000 dollars). There remains the alternative of arranging, by means of suitable switching hardware, multiport access to global memory, so that several busses can be provided each with several (10 perhaps) processors on each.

A distributor (as in Guzman's (1981) parallel multiprocessor LISP machine) is a piece of serial hardware that controls the scheduling of processors. Some means is needed to allocate processors to data (array elements) that need service. A tidy way to achieve this would be, instead of Guzman's "grill" (a repository for LISP expressions awaiting evaluation), an "activity flag" array. These are boolean flags, set up in advance of program execution, and dynamically alterable by any processor. A flag set to true is used to point to an data cell in need of service. The hardware needed is simple and flexible: just an array of consecutively numbered boolean flags such that when a flag calls a processor its ordinal is passed as an argument to the processor, which then uses that number as an index into whatever memory structure is being used. This mechanism is then not restricted to array processing but can be used to index trees, graphs (such as line drawings) etc.. However, enough flags should be available to index large data structures (e.g. 512x512 pixel images). Exceptional conditions, on array boundaries for instance, can also be efficiently handled by activity flags simply by arranging that all flags with ordinals greater than a certain number correspond to border cells. A single test on the ordinal is then enough to alert the processor to the fact that special treatment is called for.

Rather than making the distributor programmable, and programming it to avoid neighbour interactions in a particular relaxation algorithm it would be more elegant and efficient (the distributor is already very busy) for the processors to handle that. In that case some indivisible read-write operation on global

memory must be available, to enable semaphores (Bowen and Buhr (1980)) to be implemented.

Finally, an interface to a host processor is needed. This could perhaps be confined to access to control lines and status registers that indicate the current state of the parallel machine, and access to the global memory - though not necessarily while the parallel machine is running - in order to facilitate loading of programs into the processors and transfer of data - images for example.

References.

Armstrong,J.L. (1978). Programming a parallel computer for robot vision. *Computer Journal*, 21, 215-218.

Ambler,A.P., Barrow,H.G., Brown,C M., Burstall,R.M. and Popplestone,R.J. (1975). A versatile system for computer controlled assembly. *Artificial Intelligence*, *6*, 129 - 156.

Andrews,D.P., Butcher,A.K. and Buckley,B R. (1973). Acuities for spatial management in line figures: human and ideal observers compared. *Vis. Res.* 13, 599-620.

Baker,H H. (1981). Depth from edge and intensity based stereo. *IJCAI conf. 1981*.

Ballard,D.H and Brown,C M. (1982). *Computer Vision*. Prentice-Hall.

Barnett,S. and Storey,C. (1970). Matrix methods in stability theory. *Nelson, London*.

Barrow,H.G. and Tenenbaum,J.M. (1978). Recovering intrinsic scene characteristics from images. *Computer Vision Systems*, ed. Hanson,A.R. and Riseman,E.M., Academic Press, 3-26.

Barrow,H.G. and Tenenbaum,J.M. (1981). Interpreting line drawings as three-dimensional surfaces. *AI Journal*, 17, 75-116.

Beattie,R.J. (1981). Edge detection for semantically based early visual processing. Working paper 95, Dept. AI, Edinburgh University, Edinburgh.

Bellman,R. and Dreyfus,S. (1962). *Applied dynamic programming*. Princeton Univ. Press, Princeton, U.S..

Berthod,M. (1982). Definition of a consistent labelling as a global extremum. *6th Int. Conf. Pat. Rec., Munich*, 399-402.

Binford,T.O. (1981). Inferring surfaces from images. *AI Journal*, 17,205-244.

Blake,A and Ruttledge,H. (1980). CAP4 assembler and driver for a CLIP4 emulator. *MIRU, Edinburgh University, Edinburgh.*

Blake,A. (1981). Edge growing and relaxation, in parallel. *Research memorandum MIP-R-134*, MIRU, Edinburgh University, Edinburgh.

Blake,A. (1982). Fixed point solutions of recursive operations on boolean arrays. *Computer Journal*, 25, 2, 231-233.

Blake,A. (1983a). Relaxation labelling - the principle of 'least disturbance'. *Pattern Recognition Letters*, 1, 385-391.

Blake,A. (1983b). The least disturbance principle and weak constraints. *Pattern Recognition Letters*, 1,393-399.

Bowen,B.A. and Buhr,R.J.A. (1980). *The logical design of multiple microprocessor systems*. Prentice-Hall.

Brice,C.R. and Fennema,C L. (1970) Scene analysis using regions. *Artificial Intelligence 1*, 205 - 226.

Brooks,R.A., Greiner,R. and Binford,T. (1979). The Acronym model based vision system. *IJCAI conf. 1971*, 105-113.

Brooks,R.A. (1981). Symbolic reasoning among 3-D models and 2-D images. *AI Journal*, 17,285-348.

Bruha,I. (1977). The CLIP language, *Research memorandum MIP-R-120*, MIRU, Edinburgh University, Edinburgh.

Burch,S.F., Gull,S.F. and Skilling,J. (1983). Image restoration by a powerful maximum entropy method. *Computer Vision Graphics and Image Processing*. In press.

Burstall,R.M., Collins,J.S. and Popplestone,R.J. (1971). Programming in POP-2. *University Press, Edinburgh.*

Clocksin,W. (1979). The Unix POP-2 system. Documentation, Department of Artificial Intelligence, University of Edinburgh, Edinburgh.

Cottle,R.W. and Mylander,W.C. (1970). Ritter's method for non-convex quadratic programming. In *Integer and nonlinear programming*, ed. Abadie,J., 257-283.

Courant,R. (1936). Differential and integral calculus. *Blackie, Glasgow*. vol 2, 198.

Darlington,J. and Reeve,M.J. (1981) ALICE: a multi processor reduction machine. *Proc. conf. functional programming languages and computer architecture, ACM, New York*. 65-75.

Davies,P.J. (1979). Circulant Matrices. *John Wiley and Sons, New York*. 66 and ff.

Davis,L S. and Rosenfeld,A. (1977). Curve segmentation by relaxation labelling. *IEEE trans comp.*, 26, 10, 1053-1057.

Davis,L.S. and Rosenfeld,A. (1981). Cooperating processes in low level vision. *AI Journal*, 17,245-263.

Duff,M J B (1978). Review of the CLIP image processing system, *National Computer Conference 1978*, 1011 - 1060

Duff,M.J.B. (1980). Propagation in cellular logic arrays, *Proc. IEEE Workshop on Picture Data Description and Management*, 259 - 262.

Faugeras,O.D. and Berthod,M. (1981). Improving consistency and reducing ambiguity in stochastic labelling: an optimisation approach *IEEE trans. PAMI*, 3, 4, 412-424.

Feldman,J.A. and Ballard,D.H. (1982). Connectionist models and their properties. *Cognitive Science* 6, 205-254.

Feldman,J.A. and Yakimovsky,Y. (1974). Decision theory and artificial intelligence 1: a semantics based region analyser. *AI Journal*, 5, 4, 349-371.

Frisby,J.P.(1979). *Seeing*. O.U.P., Oxford.

Glaser,F. (1983). Multilevel relaxation in low-level computer vision. In *Multiresolution image processing and analysis*,
ed Rosenfeld,A.. Springer-Verlag. In press.

Gostick,R.W. (1979). Software and algorithms for the distributed-array processors. *ICL Technical Journal, May 1979*, 116 - 135.

Gottlieb,A., Grishman,R., Kruskal,C.P., McAulliffe,K.P., Rudolph,L. and Snir,M. (1983). The NYU ultracomputer - designing an MIMD shared memory parallel computer. *IEEE trans. comp.*, 32, 2, 175-189.

Guzman,A. (1981). A parallel heterarchical machine for high-level language processing. In *Languages and architectures for image processing*, ed. Duff,M.J.B and Levialdi,S. Academic Press, New York.

Hanson,A.R. and Riseman,E M. (1978) Segmentation of natural scenes. *Computer Vision Systems*, ed Hanson.A.R and Riseman,E.M., Academic Press, 129 - 164.

Hanson,A.R. and Riseman,E.M. (1978). Visions: A computer system for interpreting scenes. *Computer Vision Systems*, ed Hanson.A R and Riseman,E.M., Academic Press, 303 - 334.

Haralick,R M., Shanmugam,R. and Dinstein,I. (1973). Textural features for image classification. *IEEE trans. SMC*, 3, 610-621.

Haralick,R.M. (1980). Edge and region analysis for digital image data. *Computer Graphics and Image Processing, 12, 1*, 60 - 73.

Haralick,R.M., Mohammed,J.L. and Zucker,S.W. (1980). Compatibilities and the

fixed points of arithmetic relaxation processes. *Computer Graphics and Image Processing*, 13, 242 - 256.

Hays,W.L. and Winkler,R.L. (1970). *Statistics: probability inference and decision, vol 1.* Holt, Rinehart and Wilson, New York.

Hinton,G.H. (1978). *Relaxation and its role in vision*, Ph.D. thesis, University of Edinburgh.

Hinton,G.E. and Sejnowski,T.J. (1983). Analysing cooperative computation. *Proc. 5th conf. Cognitive Science*, Rochester, New York.

Horn,B.K.P. (1974). Determining lightness from an image. *Computer Graphics and Image Processing*, 3, 277-299.

Horn,B.K.P and Schunk,B.G. (1980). Determining optical flow. AI memo 572, AI Lab, MIT, Cambridge, USA.

Hubel,D.H. and Wiesel,T.N. (1974a). Sequence regularity and geometry of orientation columns in the monkey striate cortex. *J. Comp Neurol.* 158, 267-294.

Hubel,D.H. and Wiesel,T.N. (1974b). Striate cortex: a parallel relationship between field size, scatter and magnification factor. *J.Comp Neurol.* 158, 295-306.

Hueckel,M.H. (1971). An operator which locates edges in digitized pictures. *Journal of the Association for Computing Machinery*, 18, 1, 113 - 125.

Ikeuchi,K. and Horn,B.K.P. (1980). Numerical shape from shading and occluding contours in a single view, AI Memo 566, AI lab, MIT, Cambridge, USA.

Jelinek,J. (1979). An algebraic theory for parallel processor design, *Computer Journal, 22, no 4*, 363 - 375.

Julesz,B (1971). *Foundations of cyclopean perception.* University of Chicago Press.

Kernighan,B.W. and Ritchie,D M (1978). *The C programming language.*

Prentice-Hall.

Kushner,T., Wu,A.Y. and Rosenfeld,A. (1982). Image processing on ZMOB. *IEEE trans. comp.*, 31, 10, 943-951.

Levialdi,S., Maggiolo-Schettini,A., Napoli,M. and Uccella,G. (1981). PIXAL: a high level language for image processing. *Real Time / Parallel Computing*, Onoe,M., Preston,K. and Rosenfeld,A., Plenum Press, New York.

Lester,J.M., Brenner,J.F. and Selles,W.D. (1980). Local transforms for biomedical image analysis. *Computer Graphics and Image Processing, 13*, 17 - 30

Logan,B.T. (1977). Information in the zero crossingsof bandpass signals. *Bell Systems Tech. J.*, 56,487-510.

Luenberger,D.G. (1969). *Optimisation by vector space methods*. Wiley, New York.

Marks,P. (1980). Low-level vision using an array processor *Computer Graphics and Image Processing, 14*, 281 - 292.

Marr,D. (1976). Cooperative computation of stereo disparity. *Science*, 194, 283-287.

Marr,D , Poggio,T. and Palm,G. (1977). Analysis of a cooperative stereo algorithm. *Biol Cybernetics* 28, 223-239.

Marr,D. (1978). Representing visual information. *Computer Vision Systems*, ed. Hanson,A.R. and Riseman,E.M., Academic Press, 61 - 80.

Marr,D. and Poggio,T (1979). A computational theory of human stereo vision. *Proc. R. Soc Lond. B*, 204, 301-328.

Marr,D. and Ullman,S. (1979). Bandpass channels, zero crossings and early visual processing. *J. Opt. Soc. Am.*, 69, 914-916.

Marr,D. and Ullman,S. (1981). Directional selectivity and its use in early visual processing. *Proc R. Soc. Lond. B*, 211, 151-180.

Marr,D. (1982). *Vision.* Freeman, San Francisco, 54 ff.

Mayhew,J.E.W. and Frisby,J.P. (1981). Towards a computational and psychophysical theory of stereopsis, AI Journal, 349-385.

Mayhew,J.E.W. (1982). Personal communication.

Metropolis,N., Rosenbluth,A.W., Rosenbluth,M.N., Teller,A.H. and Teller,E. (1953). Equation of state calculations by fast computing machines. *Journal of Chemical Physics* ,6, 1087.

Montanari,U. (1971). On the optimal detection of curves in noisy pictures. *Commun. ACM* 14, 5, 335-345.

Montavo,F.S. and Weisstein,N. (1979). An empirical method that provides a basis for the organisation of relaxation labelling processes for vision. *IJCAI conf 1979*, 595-597.

Morrison,D.F. (1976). *Multivariate statistical methods.* McGraw Hill, New York, 14 ff.

Narayanan,K.A., O'Leary,D.P. and Rosenfeld,A. (1982). Image smoothing and segmentation by cost minimisation. *IEEE trans. SMC*, 12, 1, 91-96.

Nevatia,R. and Babu,K.R. (1979). Linear feature extraction and description. *CGIP*, 13, 257-269.

O'Gorman,F. (1978). Edge detection using walsh functions. *Artificial Intelligence, 10*, 215 - 223

Peleg,S. (1980). A new probabilistic relaxation scheme. *IEEE Trans. PAMI*, 2, 4, 362 - 369.

Peleg,S. and Rosenfeld,A. (1981). A note on the evaluation of probabilistic labellings. *IEEE trans SMC*, 11, 2, 176-179.

Perkins,W.A. (1978). A model-based system for industrial parts. *IEEE Trans. Comp.*, 27, 2, 126 - 143.

Poston,T. and Stewart,I. (1978). *Catastrophe theory and its applications*. Pitman, London.

Quinlan,J R. (1983). Inferno: a cautious approach to uncertain inference. *Computer Journal* 26, 3,255-269.

Ramer,E U. (1975). The transformation of photographic images into stroke arrays *IEEE Trans. CAS* ,22, 4, 363 - 374.

Reynolds,D. (1981). POPX user manual, Image Processing Group, University College, London.

Roberts,A.W. and Varberg,D.E. (1976). *Convex functions*. Academic Press, New York.

Robinson,I.N and Moore,W R. (1982). A parallel processor array architecture and its implementation in silicon. *Proc. IEEE Custom ICs conf., 1982*, 41-45.

Rosenfeld,A., Hummel,R.A. and Zucker,S.W. (1976). Scene labelling by relaxation operations. *IEEE Trans. SMC*, 6, 6, 420 - 433.

Sejnowski,T J. (1981). Skeleton filters in the brain. In *Parallel models of associative memory*. Eds Hinton,G.E. and Anderson,J A., Erlbaum, Hillsdale.

Shirai,V. (1975). Analysing intensity arrays using knowledge about scenes. In Winston,P.H. ed. *The psychology of computer vision*. McGraw Hill, New York.

Terzopoulos,D. (1982). Multilevel reconstruction of visual surfaces. Memo, 671, AI Lab, MIT, Cambridge, USA.

Ullmann J.R. (1974). A use of continuity in character recognition. *IEEE Trans. SMC*, 3, 294-300.

Ullmann J.R. (1982). Discrete optimisation by relational constraint satisfaction. *IEEE trans. PAMI*, 4, 5, 544-551.

Ullman,S. (1979). Relaxed and constrained optimisation by local processes *Computer Graphics and Image Processing*, 10, 115-125.

Varga,R. (1962). Matrix iterative analysis. *Prentice-Hall, Englewood Cliffs, NJ, USA.*

Waltz,D.L. (1972). Generating semantic descriptions from drawings of scenes with shadows. *Ph.D thesis, MIT, Boston.*

Wilkinson,J.H. (1963). Rounding errors in algebraic processes. *H.M. Stationery office.*

Wilson,H.R. and Bergen,J.R. (1979). A four mechanism model for threshold spatial vision. *Vision Research*, 19, 19-32.

Witkin,A P. (1982). Intensity based edge classification. *Proc AAAI conf 1982*, 36-41.

Zdrahal,Z. and Blake,A. (1980). A simple emulator of a parallel processor: user guide. Machine Intelligence Research Unit, Edinburgh University, Edinburgh.

Zucker,S.W., Hummel,R A. and Rosenfeld,A (1977) An application of relaxation labelling to line and curve enhancement. *IEEE trans. comp.*, 26, 4,394-403.

Zucker,S., Leclerc,Y. and Mohammed,J. (1981). Continuous relaxation and local maxima selection: conditions for equivalence. *IEEE trans. PAMI.* 3, 117-127.

Zucker,S.W (1982). Early orientation selection and grouping. Tech. rep. 82-6, McGill University, Montreal.

APPENDICES

A Fuzzy relaxation: convergence properties

Formulae involving discrete sets and set operations can be converted to formulae for fuzzy sets:

1. A set B is replaced by a fuzzy set, defined as a mapping

$$f_B: X \rightarrow [0,1]$$

assigning a probability measure for membership of the fuzzy set to each member of the universal set X.

2. Set union:

$$A \cup B = \{x : x \varepsilon A \text{ or } x \varepsilon B\}$$

corresponds to the union of fuzzy sets:

$$f_{A \cup B} = \max(f_A, f_B)$$

- itself a fuzzy set.

3. Similarly

$$f_{A \cap B} = \min(f_A, f_B)$$

4. A conversion for set complementation could be used, such as

$$f_{A'} = 1 - f_A$$

but this does not share the properties described below Discussion here will be restricted to formulae which do not contain complementation.

Now consider a threshold operator T_t which converts a fuzzy set into a discrete

set:

$$T_t(f_A)=\{x \varepsilon X : f_A(x) \geq t\}.$$

T_t also maps fuzzy union and intersection to set union and intersection:

$T_t(f_{A \cup B})=\{x : f_{A \cup B}(x) \geq t\}$

by definition of the threshold operator

$$=\{x : \max(f_A(x), f_B(x)) \geq t\}$$

by definition of fuzzy set union

$$=\{x : f_A(x) \geq t \text{ or } f_B(x) \geq t\}$$

$$=\{x : f_A \geq t\} \cup \{x : f_B \geq t\}$$

$$=T_t(f_A) \cup T_t(f_B)$$

The order in which union and thresholding are applied is interchangeable. A similar result holds for set intersection.

A formula involving sets A,B.. , with union and intersection (but *not* complementation) can be converted into a formula involving fuzzy sets $f_A, f_B \ldots$, max and min. The threshold operator then converts the fuzzy expression back to the original set formula but with $T_t(f_A), T_t(f_B)$. in place of A,B...

One last result - that a fuzzy set f_A can be reconstructed from a family of sets, S, where

$$S=\{T_t(f_A) : 0 \leq t \leq 1\}$$

by the formula:

$$\forall x \; f_A(x)=\sup_t \{t \varepsilon [0,1] : x \varepsilon T_t(f_A)\}.$$

Suppose we now convert a convergent discrete set formula D, involving sets A,B,.. union and intersection, into a fuzzy set formula, in the manner described above. This conversion was done, for instance, to convert the discrete relaxation scheme of section 2.2 into the fuzzy scheme of 2.3. We have shown that the application of the fuzzy formula is equivalent to a family of applications of the discrete formula The family of discrete formulae is:

$$\{D(T_t(f_A), T_t(f_B), \cdots) \text{ for } t \varepsilon [0,1]\}$$

where each member of the family converges in a finite time - after C_t iterations. The fuzzy formula must then converge after a time

$$c = \sup_{t \in [0,1]} C_t$$

$$= C_{t'} \text{ for some } t' \in [0,1]$$

since [0,1] is closed The convergence of the fuzzy formula in a *finite* time is guaranteed.

B Stability of non-linear relaxation.

In a non-linear relaxation scheme, as described in section 2.5, probabilities

$$p_i^{(n)}(\lambda)$$

for label λ on node i of the graph, at the nth iteration are updated according to the non-linear formula:

$$p_i^{(n+1)}(\lambda)=\frac{Q_i^{(n)}(\lambda)p_i^{(n)}(\lambda)}{\sum_{\beta\epsilon\Lambda} Q_i^{(n)}(\beta)p_i^{(n)}(\beta)} \tag{B.1}$$

where

$$Q_i^{(n)}(\lambda)=\sum_{ju} t_{ij}(\lambda,\mu)p_j^{(n)}(\mu) \tag{B.2}$$

and t is a matrix of compatibility coefficients.

Proofs are given here of the stability properties of the fixed points of non-linear relaxation.

B.1 Conditions for stability of unambiguous fixed points

To determine the stability of an unambiguous fixed point, the effect of a perturbation about the fixed point is studied. The unambiguous fixed point is:

$$p_i(\lambda) \text{ where } \forall i\ \exists\lambda_i \text{ s.t. } p_i(\lambda)=\begin{cases}1 \text{ when } \lambda=\lambda_i\\ 0 \text{ otherwise}\end{cases} \tag{B.3}$$

The perturbed labelling, at the nth iteration, is

$$p_i^{(n)}(\lambda)=p_i(\lambda)-\varepsilon_i^{(n)}(\lambda) \tag{B.4}$$

and it is clear from (B.3) that we must have

$$\varepsilon_i^{(n)}(\lambda)\begin{cases}\leq 0 \text{ when } \lambda\neq\lambda_i\\ \geq 0 \text{ when } \lambda=\lambda_i\end{cases} \tag{B.5}$$

It is sufficient to prove the stability of

$$p_i^{(n)}(\lambda) \text{ for } \lambda=\lambda_i$$

because the constraints on label probabilities

$$\forall i,\lambda\ p_i^{(n)}(\lambda)\varepsilon[0,1] \text{ and } \forall i\ \sum_{\lambda} p_i(\lambda)=1$$

ensure that

$$\forall i,\ \forall \lambda \neq \lambda_i\ \ |\varepsilon_i^{(n)}(\lambda)| \leq |\varepsilon_i^{(n)}(\lambda_i)|. \tag{B.6}$$

Clearly, therefore, it is sufficient to prove that

$$\forall i\ \ \varepsilon_i^{(n)}(\lambda_i) \to 0 \text{ as } n \to \infty$$

The non-linear updating formula from (B.1) is now applied to the perturbed labelling (B.4), and the values for $p_i(\lambda)$ in (B.3) are substituted to give:

$$p_i^{(n+1)}(\lambda_i) = \frac{Q_i^{(n)}(\lambda_i)(1-\varepsilon_i^{(n)}(\lambda_i))}{-\sum_{\mu \neq \lambda_i} Q_i^{(n)}(\mu)\varepsilon_i^{(n)}(\mu) + Q_i^{(n)}(\lambda_i)(1-\varepsilon_i^{(n)}(\lambda_i))}. \tag{B.7}$$

>From this an expression is derived for the displacement from the fixed point after one iteration:

$$\varepsilon_i^{(n+1)}(\lambda_i) = 1 - p_i^{(n+1)}(\lambda_i)$$

$$= \frac{-\sum_{\mu \neq \lambda_i} Q_i^{(n)}(\mu)\varepsilon_i^{(n)}(\mu)}{-\sum_{\mu \neq \lambda_i} Q_i^{(n)}(\mu)\varepsilon_i^{(n)}(\mu) + Q_i^{(n)}(\lambda_i)(1-\varepsilon_i^{(n)}(\lambda_i))}$$

and now rearranging terms in the denominator

$$= \frac{-\sum_{\mu \neq \lambda_i} Q_i^{(n)}(\mu)\varepsilon_i^{(n)}(\mu)}{Q_i^{(n)}(\lambda_i) - \sum_{\lambda \in \Lambda} Q_i^{(n)}(\lambda)\varepsilon_i^{(n)}(\lambda)}. \tag{B.8}$$

If the the Q values at the fixed point are

$$Q_i(\lambda) = \sum_{\mu,j} t_{ij}(\lambda,\mu) p_j(\mu) \tag{B.9}$$

then we can express, from (B.2),

$$Q_i^{(n)}(\lambda) = \sum_{j\mu} t_{ij}(\lambda,\mu) p_j^{(n)}(\mu)$$

$$= Q_i(\lambda) - \sum_{j,\mu} t_{ij}(\lambda,\mu)\varepsilon_j^{(n)}(\mu)$$

$$= Q_i(\lambda) + O(\varepsilon) \tag{B.10}$$

where $O(\varepsilon^{(n)})$ denotes an expression which depends to first order on $\varepsilon^{(n)}$ and

$$\varepsilon^{(n)}=\sup_{i,\lambda}\{|\varepsilon_i^{(n)}(\lambda)|\}. \tag{B.11}$$

So now substituting (B.10) into (B.8), we obtain:

$$\varepsilon_i^{(n+1)}(\lambda_i) = \frac{-\sum_{\mu\neq\lambda_i} Q_i(\mu)\varepsilon_i^{(n)}(\mu) + O((\varepsilon^{(n)})^2)}{Q_i(\lambda_i)+\sum_{\lambda} Q_i^{(n)}(\lambda)\varepsilon_i^{(n)}(\lambda) + O(\varepsilon^{(n)})}$$

but the second term in the denominator is also $O(\varepsilon^{(n)})$

$$= \frac{-\sum_{\mu\neq\lambda_i} Q_i(\mu)\varepsilon_i^{(n)}(\mu) + O((\varepsilon^{(n)})^2)}{Q_i(\lambda_i) + O(\varepsilon^{(n)})}$$

$$= -\frac{\sum_{\mu\neq\lambda_i} Q_i(\mu)\varepsilon_i^{(n)}(\mu)}{Q_i(\lambda_i)} + O((\varepsilon^n)^2). \tag{B.12}$$

We can now derive an expression for an upper bound on the new displacement from the fixed point $\varepsilon^{(n+1)}$ in terms of $\varepsilon^{(n)}$.

From (B.12) and (B.6)

$$\forall i\ \varepsilon_i^{(n+1)}(\lambda_i) \leq -\frac{\sum_{\mu\neq\lambda_i} Q_i(\mu)}{Q_i(\lambda_i)}.\ \varepsilon_i^{(n)}(\lambda_i) + O((\varepsilon^{(n)})^2)$$

$$\leq \frac{\sum_{\mu\neq\lambda} Q_i(\mu)}{Q_i(\lambda_i)}.\ \varepsilon^{(n)} + O((\varepsilon^{(n)})^2) \tag{B.13}$$

from (B.11), bearing in mind the signs of the perturbations at each label, as in (B.5). Now we can use (B.11) and (B.6) on the left-hand side of (B.13) to obtain

$$\varepsilon^{(n+1)} \leq \frac{\sum_{\mu\neq\lambda_i} Q_i(\mu)}{Q_i(\lambda_i)}\varepsilon^{(n)} + O((\varepsilon^{(n)})^2). \tag{B.14}$$

This guarantees that

$$\varepsilon^{(n+1)} \leq k\,\varepsilon^{(n)} \text{ with } k<1 \tag{B.15}$$

(for sufficiently small ε) *if*

$$\forall i \sum_{\mu \neq \lambda_i} Q_i(\mu) < Q_i(\lambda_i). \tag{B.16}$$

In this case,

$$\varepsilon^{(n)} \leq k^n \varepsilon^{(0)}$$

$$\to 0 \text{ as } n \to \infty$$

So we have shown that stable, unambiguous, fixed points *can* exist but unstable unambiguous fixed points may also exist, if (B.16) does not hold Here is an example, occurring in a two label two node problem:

The fixed point is.

$$p_1(\lambda_1)=p_2(\lambda_2)=1$$

and

$$p_1(\lambda_2)=p_2(\lambda_1)=0$$

where the labels are λ_1,λ_2 and in the perturbed labelling we set:

$$\varepsilon_1^{(n)}(\lambda_1)=\varepsilon_2^{(n)}(\lambda_2)=\varepsilon$$

and

$$\varepsilon_1^{(n)}(\lambda_2)=\varepsilon_2^{(n)}(\lambda_1)=-\varepsilon.$$

The compatibility matrix t is:

$$t_{ii}(\lambda,\mu)=0 \text{ for } i=1,2$$

and for $i \neq j$

$$t_{ij}(\lambda,\mu)=\begin{cases}2 \text{ when } \lambda=\mu \\ 1 \text{ when } \lambda \neq \mu\end{cases}$$

These values for the matrix t, and the values of the probabilities in the fixed point can be substituted into the condition (B.16) to show that it does not hold. After one iteration, we have a new perturbation, which can be expressed by replacing ε by ε' in the above, and using the formula in (B.8) it transpires that:

$$\varepsilon' = 2\varepsilon + O(\varepsilon^2).$$

The perturbation doubles in size, so the fixed point is unstable.

B.2 Instability of ambiguous fixed points.

We set up a perturbation of the fixed point labelling:

$$\forall i \; p_i^{(n)}(\lambda) = p_i(\lambda) + \varepsilon_i^{(n)}(\lambda) \; with \sum_{\lambda} \varepsilon_i^{(n)}(\lambda) = 0. \tag{B.17}$$

The fixed point, being ambiguous must, as we saw in section 2.5 eq (2.12), satisfy

$$\forall \lambda, i \; \sum_{\mu,j} t_{ij}(\lambda,\mu) p_j(\mu) = q_i, \tag{B.18}$$

where the q_i is some set of constants. Without loss of generality, we can replace

$$t_{ij}(\lambda,\mu) \; by \; \frac{t_{ij}(\lambda,\mu)}{q_i}$$

so that

$$\forall \lambda, \mu \; \sum_{\mu,j} t_{ij}(\lambda,\mu) p_j(\mu) = 1 \tag{B.19}$$

because the effect of this normalised version of T in the iteration formula (B.1) and (B.2) is unaffected. Equation (B.2) now becomes, in terms of the perturbation,

$$\begin{aligned} Q_i^{(n)}(\lambda) &= \sum_{j\mu} t_{ij}(\lambda,\mu) \; (p_j(\mu) + \varepsilon_j^{(n)}(\mu) \;) \\ &= 1 + \sum_{j\mu} t_{ij}(\lambda,\mu) \varepsilon_j^{(n)}(\mu) \end{aligned} \tag{B.20}$$

from (B 19). The iteration formula (B.1) can now be expanded in terms of the perturbation:

$$p_i^{(n+1)}(\lambda) = \frac{p_i^{(n)}(\lambda)(1 + \sum_{j,\mu} t_{ij}(\lambda,\mu)\varepsilon_j^{(n)}(\mu))}{\sum_{\beta} p_i^{(n)}(\beta)(1 + \sum_{j,\mu} t_{ij}(\beta,\mu)\varepsilon_j^{(n)}(\mu))}$$

which, as $\sum_{\beta} p_i^{(n)}(\beta) = 1$,

$$= \frac{p_i^{(n)}(\lambda)(1 + \sum_{j,\mu} t_{ij}(\lambda,\mu)\varepsilon_j^{(n)}(\mu))}{1 + \sum_{\beta} (\sum_{j,\mu} t_{ij}(\beta,\mu)\varepsilon_j^{(n)}(\mu)) p_i^{(n)}(\beta)}.$$

Now, using a binomial expansion of the denominator to first order in ε^n ($\varepsilon^{(n)}$ was defined in (B.10)), we obtain

$$p_i^{(n+1)}(\lambda)=p_i^{(n)}(\lambda)\ (1+\sum_{j\mu} t_{ij}(\lambda,\mu)\varepsilon_j^{(n)}(\mu)\ ).$$

$$(\ 1-\sum_{\beta}\sum_{j\mu} t_{ij}(\beta,\mu)\varepsilon_j^{(n)}(\mu)p_i^{(n)}(\beta)\ ) + O((\varepsilon^{(n)})^2)$$

and now, using (B.17), and discarding the terms which are $O((\varepsilon^{(n)})^2)$,

$$=p_i(\lambda) + \varepsilon_i^{(n)}(\lambda) + p_i(\lambda)\sum_{j\mu} \varepsilon_j^{(n)}(\mu)t_{ij}(\lambda,\mu)$$

$$- p_i(\lambda)\sum_{\beta}\sum_{j\mu} t_{ij}(\beta,\mu)\varepsilon_j^{(n)}(\mu)p_i(\beta) + O((\varepsilon^{(n)})^2) \qquad \text{(B.21)}$$

From (B.17) and (B.21) we obtain an expression for

the $\varepsilon_i^{(n+1)}(\lambda)$ *in terms of the* $\varepsilon_i^{(n)}(\lambda)$ *:*

$$\varepsilon_i^{(n+1)}(\lambda)=\varepsilon_i^{(n)}(\lambda) + p_i(\lambda)\sum_{j\mu} \varepsilon_j^{(n)}(\mu)t_{ij}(\lambda,\mu)$$

$$- p_i(\lambda)\sum_{\beta}\sum_{j\mu} t_{ij}(\beta,\mu)\varepsilon_j^{(n)}(\mu)p_i(\beta) + O((\varepsilon^{(n)})^2). \qquad \text{(B.22)}$$

We have expressed

$$\varepsilon_i^{(n+1)}(\lambda) \textit{ linearly, in terms of } \varepsilon_i^{(n)}(\lambda).$$

There is a matrix M such that

$$\varepsilon_i^{(n+1)}(\lambda)=\sum_{j,\mu} m_{ij}(\lambda,\mu)\varepsilon_j^{(n)}(\mu) \qquad \text{(B.23)}$$

where, from (B.22),

$$m_{ij}(\lambda,\mu)=\delta_{ij}\delta(\lambda,\mu) + (t_{ij}(\lambda,\mu) - \sum_{\beta} t_{ij}(\beta,\mu)p_i(\beta))\ p_i(\lambda) \qquad \text{(B.24)}$$

- δ here refers to the Kronecker δ:

$$\delta_{ij}=\begin{cases} 1 \textit{ if } i=j \\ 0 \textit{ otherwise} \end{cases}$$

and similarly

$$\delta(\lambda,\mu)=\begin{cases} 1 \textit{ if } \lambda=\mu \\ 0 \textit{ otherwise} \end{cases}.$$

The matrix M determines whether the fixed point is stable or not. If the spectral radius of M, $\rho(M)$ satisfies

$$\rho(M)>1 \tag{B.25}$$

then there is instability (Barnett and Storey 1970). Now the definition of the spectral radius is:

$$\rho(M)=\max_{k}(h_k), \tag{B.26}$$

where h_k are the eigenvectors of M. There is also a theorem (Barnett and Storey 1970, p 20) relating the eigenvectors of a matrix to its trace, and in the current context it is expressed:

$$\sum_{k} h_k = \sum_{i,\lambda} m_{ii}(\lambda,\lambda). \tag{B.27}$$

Note: eigenvalues of (geometric) multiplicity g are counted g times in the summation. Combining (B.27) and (B.26) gives:

$$\rho(M) \geq tr(M)/NL. \tag{B.28}$$

(There are N nodes, L labels, and hence the summation in (B.28) is over NL eigenvalues.) It is sufficient to investigate the trace of M. From (B.24)

$$\begin{aligned} tr(M) &= m_{ii}(\lambda,\lambda) \\ &= 1 + \left(t_{ii}(\lambda,\lambda) - \sum_{\beta} t_{ii}(\beta,\lambda) p_i(\beta) \right) p_i(\lambda). \end{aligned} \tag{B.29}$$

The trace is now

$$tr(M) = NL + \sum_{i,\lambda} F_i(\lambda) \tag{B.30}$$

where

$$F_i(\lambda) = \left(t_{ii}(\lambda,\lambda) - \sum_{\beta} t_{ii}(\beta,\lambda) p_i(\beta) \right) p_i(\lambda). \tag{B.31}$$

Now, in Peleg's probabilistic scheme,

$$t_{ii}(\lambda,\mu) \begin{cases} =0 \text{ if } \lambda \neq \mu \\ >0 \text{ if } \lambda = \mu \end{cases}$$

so that (B.31) becomes

$$F_i(\lambda) = t_{ii}(\lambda,\lambda)(1-p_i(\lambda))p_i(\lambda).$$

By the initial assumption that the fixed point is ambiguous,

$$p_i(\lambda) \neq 0 \text{ or } 1$$

so that

$$\forall i, \lambda \; F_i(\lambda) > 0$$

Hence, from (B.30), we have

$$tr(M) > NL$$

and finally, from (B.28),

$$\rho(M) > 1. \tag{B.32}$$

The fixed point is unstable. Similarly in the scheme of Rosenfeld et al. it can be shown, in the same way, that instability occurs.

Finally, if the fixed point is a mixture of ambiguously and unambiguously labelled nodes then the preceeding analysis can still be used. A perturbation can be considered in which

$$\varepsilon_i^{(n)} \neq 0$$

only if i is an ambiguously labelled node. The analysis is then restricted to these nodes, but is otherwise the same. Such a mixed fixed point is therefore unstable.

C Relaxation under linear constraints: some proofs.

C.1 Derivation of the relaxation formula.

In section 5.1 we consider a set of labels $\{X_{i\lambda}, 1\leq i\leq n, \lambda\varepsilon\Lambda\}$, under constraints S=0, where

$$S_i=\sum_{\lambda} C_{ij\lambda}.X_{j\lambda} \tag{C.1}$$

and for each $\lambda\varepsilon\Lambda$, $C_{ij\lambda}, 1\leq i,j\leq n$ is a $n\times n$ array. A distance measure

$$T = \sum_{\lambda} w_{\lambda}||X_{\lambda}-X_{\lambda}^{(0)}||^2 \tag{C.2}$$

is to be minimised, subject to the constraints. This can be done using Lagrange multipliers (Courant (1936)), M_i, and seeking a stationary point of

$$L = T + \sum_{k} M_k S_k. \tag{C.3}$$

The stationary point is given by:

$$\forall i,\lambda\ 0=\frac{\partial L}{\partial X_{i\lambda}} = X_{i\lambda}-X_{i\lambda}^{(0)} + \frac{1}{w_{\lambda}}\sum_{k} C_{ki\lambda}M_k$$

or, dispensing with indices for simplicity:

$$X_{\lambda} = X_{\lambda}^{(0)} + \frac{1}{w_{\lambda}}C_{\lambda}^{T}.M. \tag{C.4}$$

Combining (C.4) with (C.1) we obtain the constraints $S=0$ in the form:

$$\forall i\ 0=S = S^{(0)}+P.M \tag{C.5}$$

where

$$P=\sum_{\lambda}\frac{1}{w_{\lambda}}C_{\lambda}.C_{\lambda}^{T} \tag{C.6}$$

and

$$S^{(0)} = \sum_{\lambda} C_{\lambda} . X_{\lambda}^{(0)}. \tag{C.7}$$

Equation (C.5) can be regarded as an equation for M and can be solved by Jacobi relaxation (Varga 1962) as the limit of a sequence $M^{(n)}$ given by:

$$M^{(n+1)} = D^{-1} . E . M^{(n)} - D^{-1} . S^{(0)}, \tag{C.8}$$

where D is the diagonal part of P, and

$$P = D - E. \tag{C.9}$$

Having obtained this solution for M, it can be substituted back into (C.4) to obtain solutions for X_{λ}. Alternatively a sequence $X_{\lambda}^{(n)}$ can be defined by substituting $M^{(n)}$ for M in (C.4), to obtain.

$$X_{\lambda}^{(n)} = X_{\lambda}^{(0)} + \frac{1}{w_{\lambda}} C_{\lambda}^{T} . M^{(n)} \tag{C.10}$$

and therefore

$$X_{\lambda}^{(n+1)} - X_{\lambda}^{(n)} = \frac{1}{w_{\lambda}} C_{\lambda}^{T} . (M^{(n+1)} - M^{(n)})$$

$$= \frac{1}{w_{\lambda}} C_{\lambda}^{T} . D^{-1} \left[-(D-E) . M^{(n)} - S^{(0)} \right]$$

(from (C.8))

$$= -\frac{1}{w_{\lambda}} C_{\lambda}^{T} D^{-1} \left[S^{(0)} + P . M^{(n)} \right] \tag{C.11}$$

from (C.9). But if we define

$$S^{(n)} = \sum_{\lambda} C_{\lambda} . X_{\lambda}^{(n)} \tag{C.12}$$

then by following similar reasoning to that used in deriving (C.5) we obtain a similar result:

$$S^{(n)} = S^{(0)} + P . M^{(n)}. \tag{C.13}$$

So now from (C.11) and (C.13) the iteration formula can be concisely expressed as:

$$X_{\lambda}^{(n+1)} = X_{\lambda}^{(n)} - \frac{1}{w_{\lambda}} C_{\lambda}^{T} . D^{-1} . S^{(n)}. \tag{C.14}$$

This formula converges if (C.8) converges, for which a sufficient condition is that P be strictly or irreducibly diagonally dominant (Varga 1962).

C.2 Alternative derivation by vector space methods

We can regard the space of labels $X_{i\lambda}$ as a vector space of dimension $|\Lambda| \times n$. By the projection theorem (Luenberger (1969)) the constrained minimisation of T is equivalent to finding the projection of $X_\lambda^{(0)}$ onto the subspace U of consistent labellings. From the previous section, this space is

$$U = \{X_\lambda : \sum_\lambda C_\lambda . X_\lambda = 0\}. \tag{C.15}$$

Without loss of generality we take $\forall\lambda\ w_\lambda = 1$, and rescale the units of X_λ appropriately. If X_λ is the optimal labelling then, by definition of projection,

$$X_\lambda - X_\lambda^{(0)} \text{ is perpendicular to } U$$

It can be shown (Luenberger (1969, p65)) that we can therefore express this vector as

$$X_\lambda - X_\lambda^{(0)} = C_\lambda^T . M \tag{C.16}$$

where M is an n-dimensional vector. This is simply a restatement of the earlier result (C.4).

C.3 Specialisation of the derivation to edge-region duality

The edge-region duality problem of chapter 5 is to minimise

$$T = \|H - H^{(0)}\|^2 + \|V - V^{(0)}\|^2 \tag{C.17}$$

$$\text{subject to } S = 0, \tag{C.18}$$

$$\text{where } S = A.H + B.V, \tag{C.19}$$

and subject to

$$H_{ij} = 0 \text{ for } (i,j) \varepsilon Z_H \text{ and } V_{ij} = 0 \text{ for } (i,j) \varepsilon Z_V. \tag{C.20}$$

where A,B are linear mappings (convolutions, in fact, representing sparse matrices of size $(NxN)x(NxN)$), and Z_H and Z_V are sets of inactive edges. T is a convex function of (H,V) and the constraints are linear, so it is sufficient to find a local minimum of T. This is also an unconstrained local minimum of the Lagrangian, L given by:

$$L = T + 2M \times S \tag{C.21}$$

where M is an array, M_{ij}, of lagrange multipliers and × denotes direct product:

$$(M \times S)_{ij} = M_{ij} S_{ij}.$$

Setting to zero the partial derivatives of L with respect to active edges, shows that the local minimum is achieved by:

$$H = H^{(0)} + \bar{A}.M \tag{C.22}$$

where

$$\bar{A}_{i,j,k,l} = \begin{matrix} 0 \; if \; (i,j) \varepsilon Z_H \\ A_{k,l,i,j} \; otherwise \end{matrix} \tag{C.23}$$

and similarly for V, $\bar{B}$. Combining (C.18) and (C.19) with (C.22) gives

$$0 = S = S^{(0)} + P.M \tag{C.24}$$

where

$$S^{(0)} = A.H^{(0)} + B.V^{(0)}$$

and

$$P = A.\bar{A} + B.\bar{B}. \tag{C.25}$$

We reach M as the fixed point of a sequence, using a modified form of Jacobi relaxation:

$$M^{(n+1)} = Q.M^{(n)} - (1-k)D^{-1}.S^{(0)} \tag{C.26}$$

where

$$Q = kI + (1-k)D^{-1}.E, \tag{C.27}$$

I is the identity matrix, D is the diagonal part of P, and $P = D - E$. We define $S^{(n)}$ by substituting $S^{(n)}$ for S and $M^{(n)}$ for M, in (C.24), so the corresponding iteration for S is

$$S^{(n+1)} = S^{(0)} + P.M^{(n+1)} \tag{C.28}$$

$$= S^{(0)} + P.(Q.M^{(n)} - (1-k)D^{-1}.S^{(0)})$$

from eq (C.26). But we can show, from eq (C.27), that P commutes with Q and it also commutes with with D^{-1}, so now:

$$S^{(n+1)} = S^{(0)} + Q.P.M^{(n)} - (1-k)D^{-1}.P.S^{(0)}$$

$$= Q.P.M^{(n)} + (kI + (1-k)D^{-1}.E).S^{(0)} \; since \; P = D - E$$

$$= Q.P.M^{(n)} + Q.S^{(0)}$$

from eq (C.27),

$$= Q.S^{(n)}, \tag{C.29}$$

from eq (C.24). Similarly, (C.22) can be combined with (C.26) to derive a formula for $(H^{(n)}, V^{(n)})$:

$$H^{(n+1)} = H^{(n)} + (1-k)A.D^{-1}.S^{(n)} \text{ and similarly for } V. \tag{C.30}$$

It is sufficient to investigate the convergence of (C.29). When P is strictly diagonally dominant, the Jacobi scheme ($k=0$) converges (Varga (1962)). For the edge relaxation scheme, it transpires that dominance is not strict and we prove convergence for $0<k<1$. We represent A, B, P and Q as convolutions that vary over the image array. The convolutions A and B are respectively:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{bmatrix} \text{ and } \begin{bmatrix} 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

Now we can use the definitions of $\overline{A}$, $\overline{B}$, P and Q to show that in regions of active edges, the convolutions representing P and Q are respectively:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

- a laplacian operator - and

$$\begin{bmatrix} 0 & \frac{1}{4}(1-k) & 0 \\ \frac{1}{4}(1-k) & k & \frac{1}{4}(1-k) \\ 0 & \frac{1}{4}(1-k) & 0 \end{bmatrix}$$

and elsewhere the effect of inactive edges is to set one or more off-diagonal coefficients to zero, to give, for example:

$$\begin{bmatrix} 0 & -1 & 0 \\ 0 & 3 & -1 \\ 0 & -1 & 0 \end{bmatrix} \text{ and } \begin{bmatrix} 0 & \frac{1}{3}(1-k) & 0 \\ 0 & k & \frac{1}{3}(1-k) \\ 0 & \frac{1}{3}(1-k) & 0 \end{bmatrix}$$

Note that the centre coefficient changes to maintain a zero sum of coefficients.

The proof of convergence, which relies on a number of results about non-negative matrices (Varga (1962)), in outline proceeds as follows:

[1] When all the elements of H, V that contribute to an element $S^{(n)}_{ij}$ in eq (C.19) are inactive, then $S^{(n)}_{ij}=0$. Elsewhere Q is partitioned into irreducible submatrices, corresponding to the nodes of maximal sets of connected active edges (irreducible subgraphs).

[2] From (C.23) and (C.25), we can show that P is symmetric. Hence Q is symmetric, from (C.27). Each submatrix of Q is therefore symmetric, with real eigenvalues. From the specification of Q above, as a convolution, we see that

$$\forall ij \sum_{kl} Q_{ij,kl}=1 \qquad \text{(C.31)}$$

so from Varga (1962, p30,31) each submatrix will have spectral radius 1, and 1 itself is a simple eigenvalue. The eigenvalues of Q must lie in the range [-1,1]; in particular, this true when $k=0$, so that from (C.27), it follows that for $0<k<1$, the eigenvalues lie in the range (-1,1]. By inspection, the vector that is is uniform throughout the subgraph is an eigenvector of eigenvalue 1. We have seen that 1 is a simple eigenvalue, so this is the only eigenvector with eigenvalue 1. All other eigenvalues have absolute values less than 1, so as $n\to\infty$ in (C.29), $S^{(n)}$ tends to a uniform value over each subgraph.

[3] It remains to show that in fact $S^{(n)}\to 0$. For any n, summing $S^{(n)}$ over a maximal subgraph of connected active edges can be shown to yield zero. This is because, in such a subgraph, each edge contributes to the sum twice: positively at the node at one end of the edge and negatively at the node at the other end. These two exactly cancel out. The sum of $S^{(n)}$ over the subgraph is zero and $S^{(n)}$ is uniform over the subgraph, so it must be zero there.

C.4 Convergence of edge relaxation: a special case

We consider here the special case of a closed chain, length N, of active edges, under edge-region duality relaxation. This can be viewed as a one dimensional problem. The matrix Q (see appendix C.3) is the convolution

$$\left[\tfrac{1}{2}(1-k)\ k\ \tfrac{1}{2}(1-k)\right],\ where\ 0<1<k.$$

Because the chain is closed there are no special boundary conditions and Q is a circulant matrix (a convolution with wraparound at its boundaries - or equivalently a convolution on a circle). From Davies (1979) the eigenvalues of Q, other than unity, are given by

$$\lambda_r = k + \tfrac{1}{2}(1-k)\left[\exp\left[\frac{2\pi i r}{N}\right] + \exp\left[-\frac{2\pi i r}{N}\right]\right]$$

$$= 1 - 2(1-k)\sin^2\left[\frac{\pi r}{N}\right], 0 \leq r < N. \qquad \text{(C.32)}$$

The worst case eigenvalues from the point of view of slow convergence, are those closest to 1:

$$\lambda_1 = \lambda_{N-1} = 1 - 2(1-k)\sin^2\left[\frac{\pi}{N}\right] \qquad \text{(C.33)}$$

- unless k is very close to 0, in which case $\lambda_{\frac{N}{2}}$ is the worst case. Taking $k = \tfrac{1}{2}$, so that λ_1 is the worst case, we have from (C.33)

$$\lambda_1 = 1 - \sin^2\left[\frac{\pi}{N}\right]$$

which, assuming N is large,

$$\approx \exp\left[-\frac{\pi^2}{N^2}\right]. \qquad \text{(C.34)}$$

Hence $(\lambda_1)^n$ decays to $\frac{1}{e}$ after $\frac{N^2}{\pi^2}$ iterations.

Other eigenvalues decay much faster: more than half decay to $\frac{1}{e}$ within 2 iterations, $\frac{2}{3}$ within 3 iterations and $\frac{8}{9}$ within 23 iterations. Hence the convergence of S^n, in (C.29) is initially fast, but with a slow tail.

D Some properties of the graduated non-convexity method

D.1 Suboptimality in the graduated non-convexity method

We now justify a remark made in chapter 4 to the effect that: if the convex envelope method - optimisation of F^* - fails but optimising the first few functions in a sequence of functions succeeds, and those few functions are close to F^*, then the resulting solution is close to optimal.

Suppose successive optimisation is done on a family of functions $F_t(\mathbf{x}), t=0..r$, where $\mathbf{x}=(x_1, \ldots, x_n)$, a one-dimensional array, such that:

$$F_0 \equiv F^* \text{ and } F_r \equiv F \tag{D.1}$$

and also there exists an ε such that:

$$\forall \mathbf{x}, t \;\; 0 \leq F_t(\mathbf{x}) - F_{t-1}(\mathbf{x}) \leq \varepsilon \tag{D.2}$$

- bounds on the difference between successive function in the sequence. As t increases from 0 to r, we obtain a sequence of $\mathbf{x}_t$, the local minima of the corresponding F_t. We now show that $\bar{t}\varepsilon$ is a bound on the amount (cost) by which a solution $\mathbf{x}_r = \mathbf{x}_{\bar{t}}$ (i.e. no further adjustment is made after $F_{\bar{t}}$ has been dealt with), produced by the graduated non-convexity method, is suboptimal, relative to the true global optimum $\mathbf{x}'$ of F.

The graduated non-convexity method comprises a sequence of hill climbing minimisations over the functions F_t, each starting from $\mathbf{x}_{t-1}$ and ending at the local minimum $\mathbf{x}_t$. It is a property of the method that the solution $\mathbf{x}_{\bar{t}}$ satisfies

$$F(\mathbf{x}_{\bar{t}}) = F_{\bar{t}}(\mathbf{x}_{\bar{t}}). \tag{D.3}$$

Now, from (D.2),

$$\forall t \;\; F_t(\mathbf{x}_{t-1}) \leq \varepsilon + F_{t-1}(\mathbf{x}_{t-1}) \tag{D.4}$$

and also, as a property of the hill-climbing process,

$$\forall t \;\; F_t(\mathbf{x}_t) \leq F_t(\mathbf{x}_{t-1}). \tag{D.5}$$

From (D.4) and (D.5), and by a trivial induction, we obtain

$$\forall t \; F_t(\mathbf{x}_t) \leq t\varepsilon + F_0(\mathbf{x}_0). \tag{D.6}$$

But, from (D.1) and (D.2) we know that

$$F_0(\mathbf{x}') \leq F_r(\mathbf{x}') = F(\mathbf{x}'), \tag{D.7}$$

and since $\mathbf{x}_0$ is the *global* minimum of F_0

$$F_0(\mathbf{x}_0) \leq F_0(\mathbf{x}') \leq F(\mathbf{x}'). \tag{D.8}$$

Finally, from (D 6) and (D.8) we have

$$\forall t \; F_t(\mathbf{x}_t) \leq t\varepsilon + F(\mathbf{x}') \tag{D.9}$$

and, since the condition (D.3) holds for $\bar{t}$,

$$F(\mathbf{x}_{\bar{t}}) \leq \bar{t}\varepsilon + F(\mathbf{x}'). \tag{D.10}$$

The extra cost of the suboptimal solution is bounded by $\bar{t}\varepsilon$. The sooner a solution is found (the smaller $\bar{t}$ is) the closer it is to being optimal.

In practice, it is useful also to consider the case in which a solution has almost, but not quite, been achieved on $F_{\bar{t}}$, for some $\bar{t}$ (as opposed to the situation just dealt with, in which no further adjustment is required after optimisation on $F_{\bar{t}}$). In this case the residual quantity

$$E_{\bar{t}} = \left| F_{\bar{t}}(\mathbf{x}_{\bar{t}}) - F(\mathbf{x}_r) \right| \tag{D.11}$$

is small. This means that most of the work is done while $t \leq \bar{t}$ - there is not much change in the labelling as a result of the optimisations on any $F_t, t > \bar{t}$. It immediately follows from (D.9) and (D.11) that the solution $\mathbf{x}_r$ satisfies

$$F(\mathbf{x}_r) \leq \bar{t}\varepsilon + E_{\bar{t}} + F(\mathbf{x}')$$

- it is suboptimal by, at most, $\bar{t}\varepsilon + E_{\bar{t}}$. This almost the same as $\bar{t}\varepsilon$, the bound obtained previously, provided $E_{\bar{t}}$ is small. The significance of this is that, provided *most* of the change in the labelling occurs on the first few functions in the sequence the upper bound on suboptimality is small.

Finally, it is useful to be able to estimate suboptimality retrospectively, after the algorithm has been executed. It is stated in (D.7) that $F_0(\mathbf{x}_0)$ is a lower bound on the cost $F(\mathbf{x}')$ of the global minimum of F, and hence the labelling $\mathbf{x}_r$ obtained by the algorithm is suboptimal by not more than

$$F(\mathbf{x}_r) - F_0(\mathbf{x}_0). \tag{D.12}$$

D.2 Weak constancy relaxation: correctness of a special case

We consider here the special case of a step edge in a one-dimensional, n-element array, using the graduated non-convexity method to impose weak constancy constraints. Since there is only one discontinuity in the initial data the solution to the cost minimisation is either the initial state (as, for instance in fig 4.3) or a constant labelling: $\{\mathbf{x}: \forall i\ x_i=0\}$. The value of the penalty, c^2, for breaking the constancy constraint, determines exactly which of these two states is the more favourable. If the array of values is initially $\mathbf{x}^{(0)}$ then the cost of state $\mathbf{x}$ is

$$F(\mathbf{x}) = \sum_i (x_i - x_i^{(0)})^2 + \sum_{i=1}^{n-1} c^2 \left[x_i \neq x_{i+1} \right]. \qquad \text{(D.13)}$$

Suppose the height of the step discontinuity in $\mathbf{x}^{(0)}$ is $2h_0$. The cost of the initial state is, from (D.13) simply c^2 and the cost of the constant labelling is nh_0^2. The discontinuity should be flattened if the initial state has a higher cost than the constant labelling, that is iff

$$c^2 > nh_0^2, \text{ or } h_0 < \frac{c}{\sqrt{n}}. \qquad \text{(D.14)}$$

Now let us consider the effect of optimising the convex envelope function F^*, in place of the true cost function F. In chapter 4 we saw that

$$F^*(\mathbf{x}) = \sum_i (x_i - x_i^{(0)})^2 + \sum_{i=1}^{n-1} g^*\left(\frac{x_i - x_{i+1}}{a}\right) \qquad \text{(D.15)}$$

where, in the case of a linear array, $a=2$, and the function g^* was defined in chapter 4 to be

$$g^*(u) = \begin{cases} 2c\,|u| - u^2 & \text{if } |u| < c \\ c^2 & \text{otherwise} \end{cases}. \qquad \text{(D.16)}$$

To gain some physical insight into this problem, think of the first term of F as a restoring force (the x_i are on springs) towards the initial position $\mathbf{x}^{(0)}$. The second term is a set of attractive forces between pairs of adjacent x_i; these forces are not spring-like but decrease in magnitude with increasing distance between members of the pair. It can be proved (but we omit the proof here) that the stable equilibrium under these forces (unique because F^* is convex) has the form:

$$x_i = \begin{cases} -h & \text{if } 1 \leq i \leq m \\ h & \text{if } m < i \leq n \end{cases} \text{ where } m = \frac{n}{2}, \qquad \text{(D.17)}$$

for some h. The cost can then be expressed as

$$F^*(h) = n(h-h_0)^2+2ch-h^2 \tag{D.18}$$

- provided $2h<c$. Then

$$\frac{\partial F^*}{\partial h} = 2n(h-h_0)-2(h-c)$$

$$= 2[(n-1)h-nh_0+c]. \tag{D.19}$$

The position $h=0$ is stable provided

$$\frac{\partial F^*}{\partial h}\Big|_{h=0} > 0, \tag{D.20}$$

that is, provided

$$h_0<\frac{c}{n}. \tag{D.21}$$

This is the condition for the convex envelope method to flatten the step edge and should be compared with the correct solution (D.14) and it is clear that this condition (D.21) is too strong.

Now we examine the effect of the method of graduated non-convexity. The cost function F is replaced by a family of functions $\{F^{(p)}\}$, where p is a continuous parameter, $p\varepsilon[0,1]$, $F^{(0)}$ is the true cost function, and $F^{(1)}$ is the convex-envelope cost function F^*. From chapter 4, the $F^{(p)}$ are defined to be:

$$F^{(p)} = \sum(x_i-x_i^{(0)})^2+\sum g^{(p)}\left(\frac{x_i-x_{i+1}}{2}\right) \tag{D.22}$$

where

$$g^{(p)}(u) = \begin{cases} -u^2+\left[c\left(\frac{1}{p}+p\right)\right]|u| \ \textit{if} \ |u|<cp \\ c^2 \ \textit{otherwise} \end{cases} \tag{D.23}$$

As before, without proof, we will investigate only states of the form (D.17), subject to variations in h. We consider a family $\{h_p\}$ of stable positions (local minima of $F^{(p)}$). From (D.22) and (D.23) it is apparent that the influence of the force between the centre pair of neighbours, x_m and x_{m+1}, is maintained provided

$$\left|x_m-x_{m+1}\right|<cp,$$

that is, provided

$$h_p < cp. \tag{D.24}$$

If this condition is satisfied by h_p, $\forall p$ then as $p \to 0$ and the strength of the force increases, so also $h_p \to 0$. The edge is flattened.

We require to know when this condition will be met. First we calculate the h_p, assuming the force *is* still active. At equilibrium,

$$0 = \frac{\partial F^{(p)}}{\partial h}\Big|_{h=h_p}$$

$$= 2n(h_p - h_0) + c\left[\frac{1}{p} + p\right] - 2h_p$$

$$\Rightarrow 2(n-1)h_p = 2nh_0 - c\left[\frac{1}{p} + p\right]. \tag{D.25}$$

The condition $\forall p\ h_p < cp$ is obeyed iff

$$\forall p\ E_p < 0 \tag{D.26}$$

where

$$E_p = 2(n-1)(h_p - cp)$$

which, from (D 26)

$$= 2nh_0 - 2(n-1)cp - c\left[\frac{1}{p} + p\right]$$

$$= 2nh_0 - (2n-1)cp - \frac{c}{p}.$$

The minimum of E_p occurs at $p = (2n-1)^{-\frac{1}{2}}$, and is

$$E_{\min} = 2nh_0 - 2c(2n-1)^{\frac{1}{2}}. \tag{D.27}$$

The condition (D.26) is equivalent to $E_{\min} < 0$ which is satisfied iff

$$h_0 < c\left[\frac{(2n-1)^{\frac{1}{2}}}{n}\right] \tag{D.28}$$

$$\approx \sqrt{2}\frac{c}{\sqrt{n}} \text{ for large } n, \tag{D.29}$$

and this compares closely with the correct result (D.14), differing by a factor of $\sqrt{2}$. The dependence on n is therefore correct, but the effective penalty, c, is increased by $\sqrt{2}$.

D.3 Smoothness requirements for the cost function

A function $G(\mathbf{x})$ is to be minimised using a hill-climbing strategy in which only one component x_i of $\mathbf{x}$ is altered at a time. As was mentioned in chapter 4, it is necessary for G to satisfy some sort of smoothness condition, for instance that it is differentiable:

$$\forall \mathbf{x}\ G(\mathbf{x}+\delta) = G(\mathbf{x})+\nabla G(\mathbf{x}).\delta+o(||\delta||). \tag{D.30}$$

In this case, provided $||\delta||$ is small enough, the gradient vector ∇G contains all necessary information about the local slope of G; it is sufficient to investigate the components $(\nabla G)_i$ one at a time - that is, to investigate the change when $x_i \rightarrow x_i+\delta$:

$$\Delta G = (\nabla G)_i \delta+o(|\delta|). \tag{D.31}$$

In practice, however integer arithmetic does not allow infinitessimal changes. Without loss of generality we take the smallest step to be $\delta=\pm 1$. Suppose $\mathbf{x}$ is not a local minimum of G so that, at $\mathbf{x}$, ∇G has at least one non-zero component - say $(\nabla G)_{ij}>0$ (in a two dimensional array, now). Then when $x_{ij} \rightarrow x_{ij}-1$, the change

$$\Delta G = -(\nabla G)_{ij}+K, \text{ some } K. \tag{D.32}$$

Now an upper bound for K can be calculated in a particular case that interests us, $G \equiv F^*$, as defined in chapter 4. The convex function F^* was defined there in terms of the neighbour interaction function g^* ($g^{(1)}$ in equation 4.27). We can express changes in g^* as:

$$g^*(u+\delta) = \frac{\partial g^*}{\partial u}\delta+k\delta^2, \tag{D.33}$$

where

$$0 \le k \le \frac{c}{d}. \tag{D.34}$$

This arises *in* the quadratic term, $\frac{c}{8d}u^2$, in $g^*(\frac{u}{\sqrt{8}})$, near $u=0$. This can be used in the definition of F^* to show that, when $x_{ij} \rightarrow x_{ij}-1$, the change

$$\Delta(F^*) = -(\nabla F^*)_{ij}+K \tag{D.35}$$

where

$$1 \le K \le \tfrac{1}{2}+\tfrac{1}{2}\frac{c}{d}. \tag{D.36}$$

The worst case is $K=\frac{1}{2}+\frac{1}{2}\frac{c}{d}$, in the sense that then, even though $-(\nabla F^*)_{ij}$ is negative, $\Delta(F^*)$ could still be positive unless

$$\left|(\nabla F^*)_{ij}\right| > \frac{1}{2}+\frac{1}{2}\frac{c}{d}. \tag{D.37}$$

Provided condition (D.37) holds, a downward slope in the function F^* is bound to be discovered by taking a unit step in x_{ij}. Exploring F^* in unit steps of $\mathbf{x}$ reveals all but small gradients; choosing a larger d enables smaller gradients to be detected.

The effect of the constant d can be understood more clearly by considering an example. A square of size $n \times n$ elements, each taking the value h_0, sits in a background of zero values. Consider now the cost $F^*(\mathbf{x})$ near this initial state $\mathbf{x}^{(0)}$. If we consider changing all the values in the square from $h=h_0$ to $h=h_0+\delta$, it can be shown that the corresponding change in cost (assuming $d<h_0<c$) is:

$$\Delta(F^*) = \sqrt{8}nc\,\delta + O(\delta^2) \tag{D.38}$$

so that, taking the limit as $\delta \to 0$,

$$\frac{\partial F^*}{\partial h} = \sqrt{8}nc. \tag{D.39}$$

But

$$\frac{\partial F^*}{\partial h} = \sum_{ij} \frac{\partial F^*}{\partial x_{ij}} \tag{D.40}$$

where $\frac{\partial F^*}{\partial x_{ij}} \equiv (\nabla F^*)_{ij}$. This is a sum of n^2 terms so that, from (D.39), there must be at least one element x_{ij} s.t.

$$\frac{\partial F^*}{\partial x_{ij}} \geq \frac{\sqrt{8}nc}{n^2} = \sqrt{8}\frac{c}{n}. \tag{D.41}$$

Applying this to the earlier result (D.37), the gradient in F^* can be detected if

$$\sqrt{8}\frac{c}{n} > \frac{1}{2}+\frac{1}{2}\frac{c}{d}$$

which, if c is large, is equivalent to

$$d > \frac{n}{4\sqrt{2}}. \tag{D.42}$$

This simple result means that the parameter d can be interpreted as a characteristic distance in the array over which the discrete hill-climbing algorithm can

act. This result is independent of c.

D.4 Constructing a convex cost function

We address here the problem of constructing a convex function $F^*(\mathbf{x})$, given a cost function $F(\mathbf{x})$, where $\mathbf{x}=(x_1, \ldots, x_n)$. In chapter 4, the requirements were that F^* should be convex, that

$$\forall \mathbf{x}\ F^*(\mathbf{x}) \leq F(\mathbf{x}) \tag{D.43}$$

and that F^* should be as close as possible to F. In general, for relaxation under weak constraints, the cost function is, initially,

$$F(\mathbf{x}) = ||\mathbf{x}-\mathbf{x}^{(0)}||^2 + c^2\sum\left[l_i^{(1)}\neq 0\right] + \cdots + c^2\sum\left[l_i^{(s)}\neq 0\right], \tag{D.44}$$

where the $l_i^{(r)}, r=1..n$ are different linear combinations of $\mathbf{x}$, expressed by convolutions:

$$l_i^{(r)} = \sum_k h_{i-k}^{(r)} x_k \tag{D.45}$$

The index $i-k$ in (D.45) is interpreted as lying on a circle of n points - $(i-k)$ *modulo* n is understood. The first step in convex function construction has been described in chapter 4, and involved replacing terms like $c^2[u\neq 0]$ by $g^*(\frac{u}{a})$, for some constant a, where g^* was defined with the property that

$$u^2+g^*(u) \text{ is convex - i.e. } \frac{\partial^2 g^*}{\partial u^2} \geq -2, \tag{D.46}$$

with equality attained by some u.

This substitution of g^* in F defines F^* to be:

$$F^*(\mathbf{x}) = ||\mathbf{x}-\mathbf{x}^{(0)}||^2 + \sum_r\sum_i g^*(\frac{l_i^{(r)}}{a}) \tag{D.47}$$

The task that remains is to determine a value of the constant a, just sufficiently large that F^* is convex. We require a to be as small as possible so that F^* is as close as possible to F while still keeping F^* convex. In fact we require F^* to be convex, but not strictly - F^* is to be a limiting case and hence must have zero curvature somewhere.

The curvature of $g^*(\frac{u}{a})$ is given by

$$\frac{\partial^2}{\partial u^2}\left[g^*\left(\frac{u}{a}\right)\right] = \frac{1}{a^2}g^{*\prime\prime}\left(\frac{u}{a}\right)$$

$$\geq -\frac{2}{a^2}, \tag{D.48}$$

from (D.47). Equivalently, the function

$$\left(\frac{u}{a}\right)^2 + g^*\left(\frac{u}{a}\right)$$

is convex (not strictly). Therefore, referring back to (D.46) and (D.47), F^* is convex iff the function

$$G(\mathbf{x}) = \tfrac{1}{2}||\mathbf{x}-\mathbf{x}^{(0)}||^2 - \tfrac{1}{2}\sum_r\sum_i\left[\frac{l_i^{(r)}}{a}\right]^2 \tag{D.49}$$

is convex. To determine this whether G is convex, we inspect the Hessian matrix, H, of G (Roberts and Varberg (1976)). It is convex iff H is positive definite

By definition,

$$H_{ij} = \frac{\partial G}{\partial x_i \partial x_j}$$

$$= \delta_{ij} - \frac{1}{a^2}\sum_r G_{ki}^{(r)} G_{kj}^{(r)}. \tag{D.50}$$

Here δ is the Kronecker δ and summation convention is used, and the

$$G_{ij}^{(r)} = \frac{\partial l_i}{\partial x_j} \tag{D.51}$$

which, from (D.45),

$$= h_i^{(r)}_j.$$

Therefore H is positive definite iff the largest eigenvalue λ_{max} of the matrix

$$\bar{h}_{ij} = \sum_r h_k^{(r)}_i h_k^{(r)}_j \tag{D.52}$$

satisfies

$$\lambda_{max} < a^2. \tag{D.53}$$

The matrix $\bar{h}$, it can be shown, is a circulant matrix (Davies (1979)) for which

λ_{max} can conveniently be calculated.

As an example, weak continuity relaxation, as described in chapter 4, has constraints represented by two convolutions, $h^{(1)}$ and $h^{(2)}$:

$$h^{(1)} \text{ is } \begin{bmatrix} 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \text{ and } h^{(2)} \text{ is } \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{bmatrix}. \tag{D.54}$$

From (D.52)

$$\bar{h} = (h^{(1)})^T . h^{(1)} + (h^{(2)})^T . h^{(2)}, \tag{D.55}$$

which is the convolution:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}. \tag{D.56}$$

Its eigenvalues are given by the formula (a variant, for two dimensional arrays, of the formula given by Davies (1978)):

$$\lambda_{rs} = 4 - 2\cos\left[\frac{2\pi r}{n}\right] - 2\cos\left[\frac{2\pi s}{n}\right], \quad 1 \le r, s \le n. \tag{D.57}$$

Hence $\lambda_{max}=8$. Therefore, from (D.53), the constant α is set to $\alpha=\sqrt{8}$.

For weak harmonic constraints, the constraints are represented by the single convolution $h^{(1)}$, which is:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}. \tag{D.58}$$

which, as we have just seen, has maximum eigenvalue 8. Because it is symmetric, the matrix

$$\bar{h} = (h^{(1)})^T . h^{(1)} = (h^{(1)})^2, \tag{D.59}$$

which has maximum eigenvalue 8^2. Hence the constant $\alpha=8$ in this case.

D.5 Parameters affecting the convergence of weak constancy relaxation.

The following tables exhibit aspects of convergence and performance for the weak constancy relaxation algorithm described in chapter 4. Execution times refer to the asynchronous/single processor implementation, described in section 4.4, running on a PDP 11/24 computer. Except where specified otherwise, parameters of the algorithm are set as follows:

Penalty for breaking constraint, C=64.

[Note that $C=8^{\frac{1}{2}}c$, where c is the penalty as specified in section 4.2. The change of scale is simply for computational convenience.]

smoothness, d=16.

no of non-convex functions: 4.

precision: 12 bits.

maximum iterative adjustment: 16.

array size: 64x64.

The meaning of these parameters is explained in section 4.2. In each table measurements are given for each of three pictures: fig F.1a, fig F.2a and fig F.3a.

TABLE 1: Effect of the cost function smoothness parameter, d. A larger value of d indicate greater smoothness and requires more computation.

	Exec. time (s)				Total iterations			
d	4	8	16	32	4	8	16	32
Teapot	260	350	420	423	749	1042	1250	1188
Spanner	220	274	292	299	605	749	710	774
Cutters	236	300	315	333	579	721	656	847

TABLE 2: Effect of the integer precision used to represent the image, measured in bits. Higher precision requires more computation.

	Exec. time (s)				Total iterations			
precision	9	10	11	12	9	10	11	12
teapot	77	135	240	420	184	358	693	1250
spanner	63	105	177	292	159	240	454	710
cutters	70	117	198	315	159	282	517	656

TABLE 3: Effect of the number of functions, n, in the sequence of non-convex cost functions. Optimisation is performed successively on each function in the sequence. The cost figure quoted in the table is the final true cost of the result image. The global minimum cost is a lower bound on the cost of the true global minimum. More functions means more computation, but a more nearly optimal result.

	Exec. time (s)				Final cost				Global
n	1	2	4	13	1	2	4	13	
teapot	161	261	420	943	1436	1333	1195	1067	760
spanner	125	190	292	717	425	377	356	361	322
cutters	143	209	315	844	950	921	1018	671	408

TABLE 4: Effect of using a coarse-to-fine progression in the size of adjustments applied to the image array. The adjustment size progresses from maximum size as specified, successively halved, down to unit size. Figures are given for the convex cost function F^* only. Clearly the coarse-to-fine strategy results in reduced computation time.

	Exec. time (s)					Total iterations				
max.ch.	1	2	4	8	16	1	2	4	8	16
teapot	250	147	104	88	83	448	247	160	132	118
spanner	197	119	86	76	75	261	172	120	113	112
cutters	238	142	101	85	81	238	142	101	85	81

TABLE 5: Effect of varying the penalty, C, for breaking a constraint. Note that 12 bit precision is used except for C=128, where precision is 11 bits. Larger values of C require more computation.

	Exec. time (s)				Total iterations			
C	16	32	64	128	16	32	64	128
teapot	176	247	420	514	379	559	1250	1314
spanner	187	235	292	399	341	519	710	1134
cutters	192	257	315	440	358	575	656	1083

TABLE 6: Effect of varying the penalty, C, for breaking a constraint, and using an "economy" regime - setting algorithm parameters for the shortest feasible execution time. Parameters are set as in the previous table except for.

precision: 10 bits. no of non-convex functions: 2. (indices 0.3 and 0.09). smoothness, d=8.

	Exec. time (s)				Total iterations			
C	16	32	64	128	16	32	64	128
teapot	48	61	88	131	77	126	216	417
spanner	51	59	67	95	76	114	141	303
cutters	52	61	78	107	85	106	178	295

TABLE 7: Effect of varying array size. For comparability, the same image is used scaled to each of three sizes, and the penalty, C, scales as the square root of the image linear dimension Optimisation is over the convex cost function F^* only. Larger arrays require more computation; execution time appears to vary roughly in proportion to array area.

	Exec. time (s)			Total iterations		
size	16x16	32x32	64x64	16x16	32x32	64x64
c	64	90	128	64	90	128
teapot	6	29	118	86	142	174
spanner	6	24	95	100	127	156
cutters	7	26	103	88	124	158

E. PARAPIC user manual

MACHINE INTELLIGENCE RESEARCH UNIT

UNIVERSITY OF EDINBURGH

Subject: Parapic language definition

Version: 2.1

Author: A. Blake

Date: April, 1982

Contents

1 Introduction

PARAPIC is a language for image processing, with two important features. First, it is for processing pictures in parallel; each primitive of the language consists of a sequence of the basic operations that are available in parallel array processors like the CLIP (Duff 1978) and the DAP (Marks 1980). The basic operations of the CLIP have been thoroughly analysed by Jelinek (1979). Secondly, it is a high level language, an extension of POP-2 (Burstall et al. 1971). PARAPIC bears some resemblance to APL (Iverson), in that there are array data-types and that arithmetic operators, applied to array variables, perform array operations. It is, however, more specifically tailored to image processing, and in that sense has design principles similar to those of PIXAL (Levialdi (1981)) and those in (Bruha 1977). PARAPIC has been implemented, running under UNIX† on PDP-11 minicomputers. It drives a parallel array processor emulator, whose design has evolved from the work of Armstrong (1978), Zdrahal and Blake (1980) and Blake and Ruttledge (1980).

There are two components of the PARAPIC extension to POP-2. The image data type is introduced and a number of new operators and functions are added. An image variable is a square array which takes one of three forms:

boolean image - an array whose cells contain truth values

grey image - an array whose cells contain 8-bit signed integers

double image - a double precision grey image, whose cells contain 16 bit signed integers

Of the new operators and functions, some are entirely new (for instance array shift) and some are arithmetic operators or functions, extended to act over arrays (for instance + - * /).

As an example of an arithmetic operation over arrays, the + operator acts as follows:

In $C=A+B$, the array C has components each of which is the sum of corresponding components in A and B The expression above is equivalent, in component form to

$$for\ all\ i,j\ C_{ij}=A_{ij}+B_{ij}$$

†UNIX is a Trademark of Bell Laboratories.

and x - / and other arithmetic operations are similarly defined for grey and double arrays. For boolean arrays there is a set of boolean or logical operations, for instance & (AND) | (OR), and they are defined over array components, similarly to the arithmetic operations. An important operator for arrays, which is not an extended arithmetic operation, is the shift operator |>. In the expression $C=A|>d$, C denotes the array A shifted in direction d. For instance, when d is North,

$$C=A|>(North) \text{ means: for all } ij \; C_{ij} = \begin{matrix} A_{i-1,j} \text{ if } i \geq 1 \\ 0 \text{ otherwise} \end{matrix}$$

Other operations available in PARAPIC include

- parallel array conditional expression; this is simply an extension to arrays of the ? operator, in which $C=B?(X,Y)$ means IF (B) THEN C=X ELSE C=Y. B must be boolean valued and X,Y must, in the array operation, share the same type.

- global measurements over boolean and grey arrays

- image input/output

- start and stop functions for the underlying image processing machine

These operations are specified in more detail in chapter 4.

The PARAPIC language is specified without reference to a particular target parallel array processor. However the choice of operations is constrained by the notion of a parallel array machine. All the operations in the language extension can be succinctly expressed in terms of the basic operations of the CLIP4 and the DAP. PARAPIC provides, therefore, a natural programming environment for the development of image processing programs which make good use of the high processing capacity of parallel array machines.

The PARAPIC language, as specified here, has been implemented by the Author, under the UNIX operating system, driving a software emulator of a parallel array processor. A sample session is shown in appendix A Software based on PARAPIC (Reynolds 1981) has been written which uses the CLIP4 machine although it is not possible, using interpreted POP-2 on the PDP-11 (Clocksin 1979), to drive the CLIP4 at its full speed.

Summary

The PARAPIC image processing language consists of the POP-2 high level language with an extension for image processing. The extension is composed of a data-type - the image - which may be integer or boolean valued, and a substantial set of image operators and functions. PARAPIC embodies the constraints imposed by the structure of the parallel array processor and facilitates the development of image processing programs, within those constraints, and in a high level language environment.

2 Image types and type conversion

Three image types are available: boolean, grey and double precision, which are arrays of truth values, eight bit signed integers and sixteen bit signed integers, respectively. The arrays are square and of a size determined by the underlying parallel array machine or software emulator. In the Unix implementation of PARAPIC (described in the appendix), for instance, a parallel array emulator is started up by the PARAPIC command *

: **start** *(size)*;

where *size* is the length of the sides of the square arrays.

Initialised image are obtained by calling the functions **initb**, **initg** and **initd**. For instance:

initb(**true**) returns a boolean image array of cells set to **true** (1).

initg(0) returns a grey image array of cells set to 0.

initd (-4000) returns a double precision array of cells set to -4000.

Functions are provided for conversion between types. The function **extend** converts a grey image to a double image with each cell having the same value (including sign) as the corresponding cell in the grey image. The functions **upper** and **lower** yield, from a double image, a grey image consisting of the most or least significant bytes, respectively, from the cells of the double image. Conversion from grey or double to boolean is achieved by arithmetic comparisons (> = etc.) described later on. Conversion from boolean to grey or double is done by the **?** operator, also described below.

* The POP-2/PARAPIC prompt is '.'.

3 Image input and output

Several functions and operators are provided for image input and output, both for transfer to and from files on the host machine, and for communication with other image processing processes - these will often be processes for image capture or display. In all cases the result of input is a grey image, and any picture submitted to PARAPIC for input must be the same size as the image arrays. Any required windowing or magnification must be performed by external processes, invoked from PARAPIC, as part of the picture input/output command. Pictures for output are 8 bit grey and of the same dimensions as the PARAPIC image arrays. A grey image may therefore be submitted unchanged for output, whereas a boolean image is first converted to grey (with one grey level representing true and another representing false - system constants), and a double image is converted to grey by the function **upper** before display. †

The input/output operators are:

<< *'filename'* and >> *'filename'*

respectively, where the single quoted string is the name of a file on the host computer Also:

<<|' *command'* and >>|' *command'*

perform input and output respectively, where the single quoted string is a name for a process (this assumes a multi process operating environment on the host machine). Under UNIX for instance, *command* is a shell command which executes a process whose input/output is piped to/from PARAPIC. The view and display operators

<<* and >>*

perform piped input and output of pictures, as described above, but the process called in each case is chosen in advance by executing

: **setview** (' *command* ')

† It is intended, in the future, to adopt a picture file format which allows boolean, double precision and unsigned images, in which case these conversions will be unnecessary.

and

setdisp (' *command* ')

respectively.

4 Image operators and functions

This section describes the image processing operators and functions available in PARAPIC, and the image types to which they may be applied.

4.1 Boolean operators

The unary operator ~ inverts a boolean image - all true cells become false and vice versa. The binary operators **| & ~| |~ ~& &~ @ ==** produce a boolean image, from two boolean image arguments The logical **or and** are performed over all the array cells by the operators **| &** ; the operators **~| |~ ~& &~** also include an inversion, for instance:

$$a|{\sim}b \equiv a|(\sim b)$$

and

$$a{\sim}\&b \equiv (\sim a)\&b$$

The exclusive or operator, @, is defined by:

$$a@b \equiv (a\&(\sim b))|((\sim a)\&b).$$

The equivalence operator, ==, is defined by:

$$a{=}{=}b \equiv \sim(a@b).$$

4.2 Grey and double operators

Operations on grey and double images are either arithmetic, yielding grey or double images, or comparisions, which yield boolean images. For the arithmetic operations with one argument, allowed types are:

```
 grey      -> grey
double   -> double
```

and with two arguments:

(grey,integer)	-> grey
(grey,grey)	-> grey
(integer,grey)	-> grey
(integer,double)	-> double
(double,double)	-> double
(double,integer)	-> double

The unary operator ~, performs ones complementation throughout the grey or double array. The binary operators + * - / perform addition, multiplication, subtraction, signed integer division and remainder. The first four of these operators are already used in POP-2 for arithmetic, so that the allowed types specified above are in addition to the types already allowed in POP-2. The function **abs**(*a*) and its alias **mod**(*a*) - modulus and the functions **max**(*a*, *b*) and **min**(*a*, *b*) - maximum and minimum - have been incorporated. The POP-2 functions **logand**(*a*, *b*), **logor**(*a*, *b*) and **logshift**(*a*, *b*) which perform bitwise **and, or, shift** on integers, are extended for grey and double images Note that the right shift for pictures *does* include sign extension. The POP-2 function **sign**(*a*) has also been extended for grey, double images.

Comparisions

The operators > >= < =< = /= are extended from the POP-2 definition to yield boolean images from grey/double. The allowed types of arguments are the same as for the arithmetic operations, with two arguments, above.

4.3 Shifting and propagation

The array shift is a basic parallel array processor function, arising from the connections between neighbouring cells in the processor array. Thus, shifting an array from the North (Northwards shift means shift *from* the North as in "North wind") each cell takes on the value of the cell immediately to its North. Cells along the Northern border of the array, assume the value 0. The shift operator, |>, in PARAPIC, is applied to a boolean, grey or double image, *p*:

p |> *dir*

in direction *dir*, which must be in the range 1..4. Directions 1,2,3,4 are North, East, South, West. The resulting image has the same type as *p*.

Some operations which can be synthesised from the shift are available in PARAPIC. The expand and shrink operators, ^^and ~^, are defined by:

$$\text{^^}a \equiv a|(a|>1)|(a|>2)|(a|>3)|(a|>4)$$

and

$$/\text{^}a \equiv a\&(a|>1)\&(a|>2)\&(a|>3)\&(a|>4)$$

The propagation function

prop(*a*, *b*, *dirstring*)

propagates a boolean image b, inside a boolean image a, in the directions specified in *dirstring*. We define

dir ::= 1|2|3|4
dirstring ::= ' *dir* *',

The function **prop** may be defined recursively as:

prop(*a*, *b*, *dirstring*) =
    IF there exists dir in dirstring such that
        (b|>dir)&a
    is not everywhere (throughout the array) identical to *a* THEN
        prop(a,(b|>dir)&a,dirstring)
    ELSE
        b

In the case, for instance, that

dirstring = '1234'

the effect of **prop** is to produce a boolean image which contains **true** at any cell that is joined by an unbroken path of **true** cells in *a*, to some true cell in *b*, sometimes referred to as 'labelling' *a* with *b* or as the 'reconstruction' of *b* in *a*.

The function

frame(*dirstring*)

returns a boolean image which contains **true** cells along those array edge specified in dirstring and **false** elsewhere. Thus

frame('1234')

returns an array with **true** cells around all four edges and **false** everywhere else.

Finally the function

ramp(*dir*)

which can also be synthesised using shift, produces a grey image containing a grey level ramp. It has 0's along the edge specified in *dir* and the cell contents increase by one grey level per pixel, towards the opposite edge. For instance:

: ramp(1) -> a;

produces an array whose components are:

$$For\ all\ j\ a_{ij} = i - 1.$$

4.4 Parallel conditional expression

The ternary parallel conditional operator, **?**, combines a boolean image with grey/double arguments to produce a grey/double image. Executing:

: b?(d,e) -> c;

produces an image c which, in component form, is:

$$c_{ij} = \begin{matrix} d_{ij}\ if\ b_{ij}\ (=true) \\ e_{ij}\ otherwise \end{matrix}$$

The argument b must be a boolean image and allowed types for d,e are exactly as for the arguments of a binary arithmetic operator (see section 4.2) and with the same type for the returned value.

4.5 Global array measures

The image operations described so far all produce images as their result. The global array measures described here, however, operate on an image but return boolean (as opposed to boolean image), integer or real results. For instance, the predicates **everywhere**, **somewhere** and **nowhere**, applied to a boolean image, b, return **true** if:

everywhere(b): all cells in b are true

somewhere(b): there exists a true cell in b

nowhere(b): all cells in b are false

and **false** otherwise The function **area** returns the number of true cells in a boolean image. The function **greymax**, **greymin**and **greyav** return the maximum, minimum and average, respectively, of all the cells in a grey or double array.

Acknowledgements

The author wishes to acknowledge the considerable contribution made to the design and implementation of PARAPIC by Hugh Ruttledge, who suggested writing a parallel image processing language, and wrote much of the software for a prototype.

The author acknowledges the contribution of W.Clocksin, in the form of his POP-2 system for Unix.

This research was conducted with the aid of a grant from the SERC to Professor Donald Michie for work on a versatile programmable industrial image processor. The author is also indebted to the University of Edinburgh for the provision of facilities.

References

Armstrong,J.L. (1978). Programming a parallel computer for robot vision. *Computer Journal*, 21, 215-218.

Bruha,I. (1977). The CLIP language, *Research memorandum MIP-R-120*, MIRU, Edinburgh University, Edinburgh.

Blake,A. and Ruttledge,H. (1980). CAP 4 assembler and driver for CLIP4 emulator Machine Intelligence Research Unit, University of Edinburgh.

Burstall,R.M., Collins,J.S. and Popplestone,R.J. (1971). Programming in POP-2. *University Press, Edinburgh.*

Clocksin,W. (1979). The Unix POP-2 system. Documentation, Department of Artificial Intelligence, University of Edinburgh, Edinburgh.

Duff,M.J.B (1978). Review of the CLIP image processing system, *National Computer Conference 1978*, 1011 - 1060.

Iverson,K. (). A Programming Language.

Jelinek (1979). An algebraic theory for parallel processor design. *Computer Journal*, 22 (4), 363-375

Levialdi,S., Maggiolo-Schettini,A., Napoli,M. and Uccella,G. (1981). PIXAL: a high level language for image processing. *Real Time / Parallel Computing*, Onoe,M., Preston,K. and Rosenfeld,A., Plenum Press, New York

Marks,P. (1980). Low-level vision using an array processor *Computer Graphics and Image Processing*, *14*, 281 - 292.

Reynolds,D. (1981). POPX user manual, Image Processing Group, University College, London

Zdrahal,Z. and Blake,A (1980). A simple emulator of a parallel processor: user guide. Machine Intelligence Research Unit, Edinburgh

Appendices

A. PARAPIC under Unix, with an array processor emulator.

To enter PARAPIC, type

parapic <RETURN>

and the usual POP-2 prompt : appears. To start the emulator, working with 64 x 64 arrays.

```
: start(64);
```

The arraysize must be a multiple of 16, between 16 and 128 inclusive. Now try some commands:

```
: ramp(4) -> g;
```

puts a grey level ramp into g.

```
: g >>* ;
```

displays it.

```
: (g >= 16) & (g < 48) -> b;
```

makes a boolean image, b, with a central vertical stripe of **true**, 32 pixels wide.

```
: b >>* ;
```

displays b. Now use b to mask off the parts of g which lie outside the stripe, to produce h.

```
: b?(g,0) -> h;
```

The result can be saved on a file called masked_ramp, by executing:

```
: h >> 'masked_ramp' ;
```

Typing control-Z stops the emulator, and exits from the PARAPIC system, returning to the shell. On a future occasion,

```
parapic

: start(64) ;

: <<'masked_ramp' -> g ;
```

retrieves the masked ramp, for further processing.

B. Summary of operators and functions.

Operator	Precedence	Description
<< <<| <<*	1	Picture input
~	2	Invert / 1's complement
|> ^^ /^	3	shift, expand, shrink
* / %	4	arithmetic
+ -	5	arithmetic
> >= < =< = /=	6	comparisons
| & |~ ~| &~ ~& @ ==	7	boolean operations
?	8	parallel conditional
>> >>| >>*	9	output

Note that precedences follow POP-2 conventions. In an expression the operator with lowest precedence is evaluated first.

Function	description
setview setdisp dump	picture input/output
start stop	image processor control
initb initg initd	image initialisation
logor logand logshift sign abs(mod) max min	arithmetic
upper lower extend	double/grey conversion
prop frame	propagation, boolean frame
ramp	grey ramp
area somewhere nowhere everywhere	global boolean measures
greymax greymin greyav *	global grey measures

* not yet implemented.

F Some results of weak constancy relaxation applied to real images

The following figures show examples of the effect of weak constancy relaxation, both on intensity and on angle data. Where "economy regime" has been used this refers to particular settings of parameters (see appendix D.5) chosen to reduce execution time. This results in some degradation of the labelling produced. In all cases, 12 bit precision is used for C=32,64, 11 bit for C=128 and 10 bit for C=256; also the smoothness constant d=16 for C=32,64,128 and d=8 for C=256. These choices of parameters are to allow computation to be done with 16-bit integers.

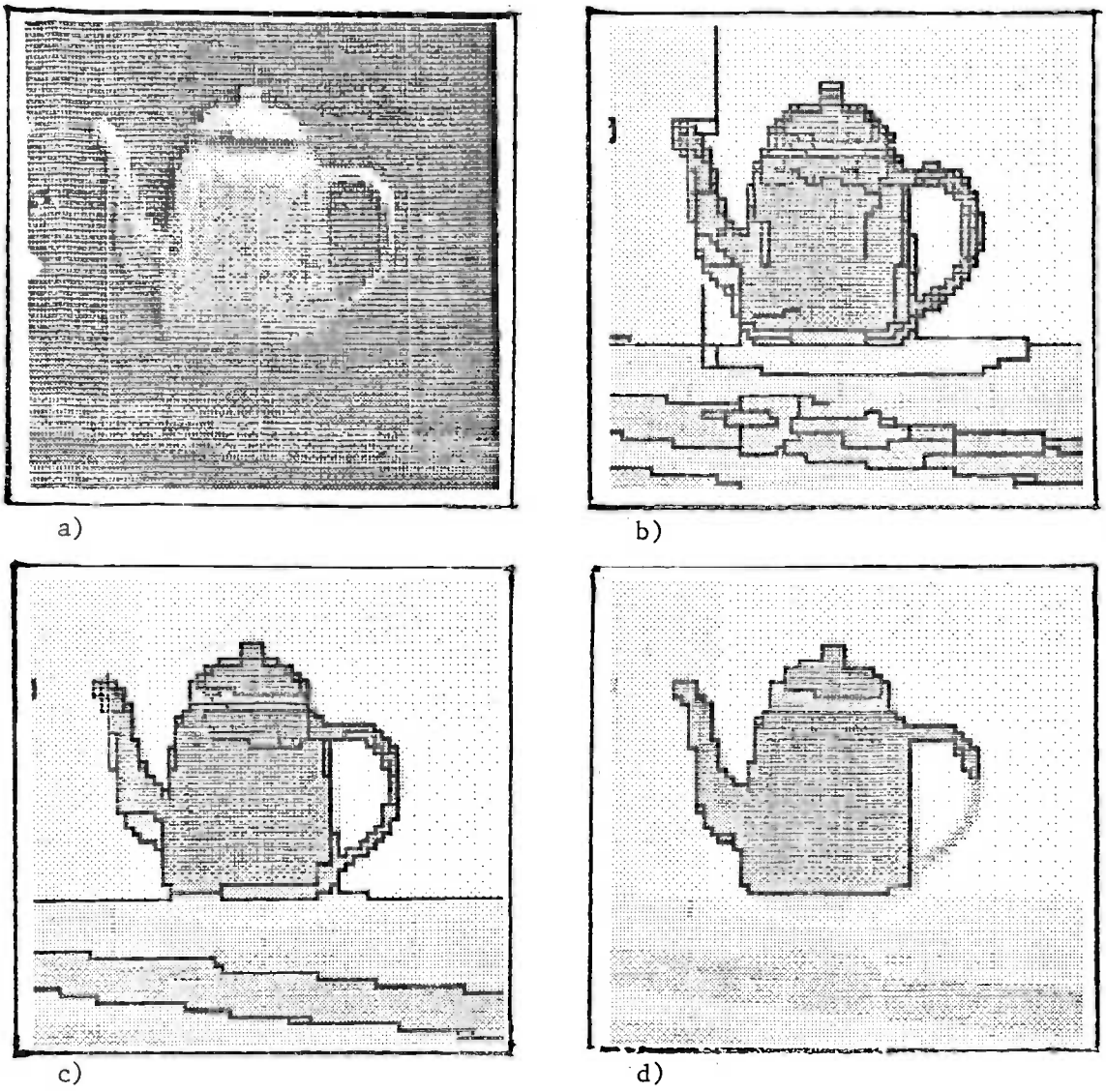
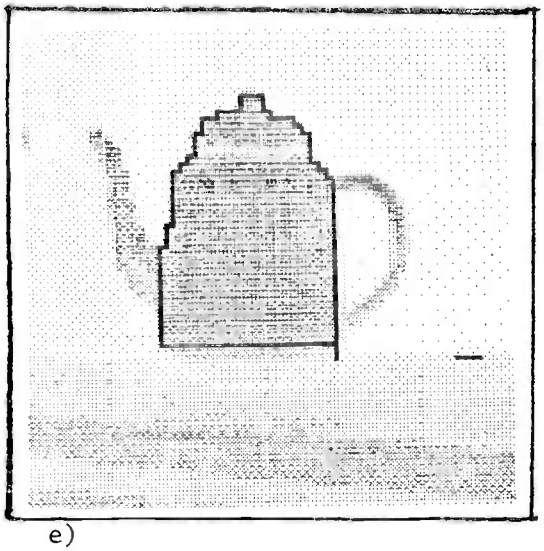
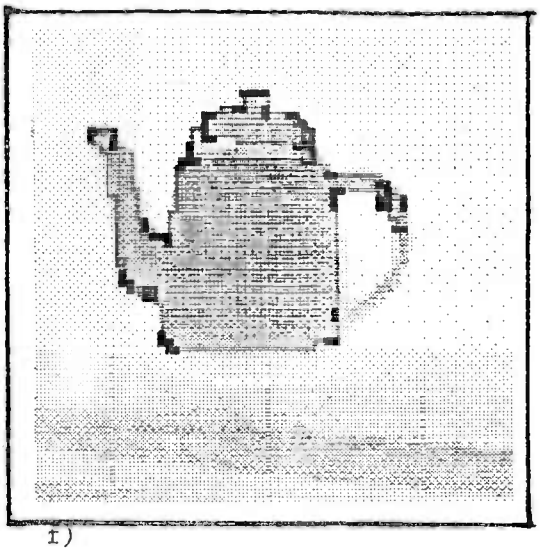


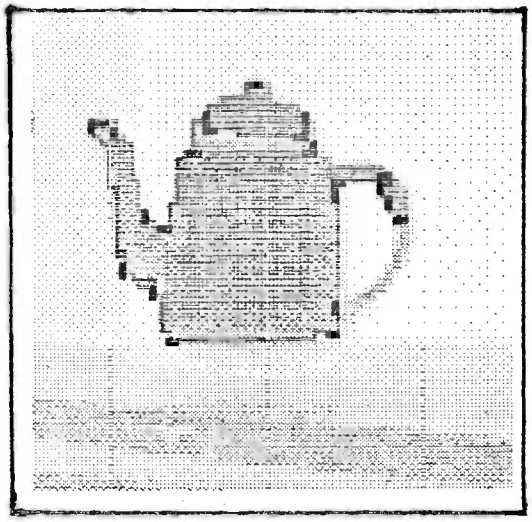
Fig F.1 Weak constancy relaxation on image (a) to define intensity discontinuities (b-e), using penalty C=32,64,128,256 respectively. Weak constancy angle relaxation: (f&g) from the lines in (d) with C=50,100 respectively; (h&i) from (e) with C=50,100. Intensity relaxation using the "economy regime" (j-m), C=32,64,128,256.



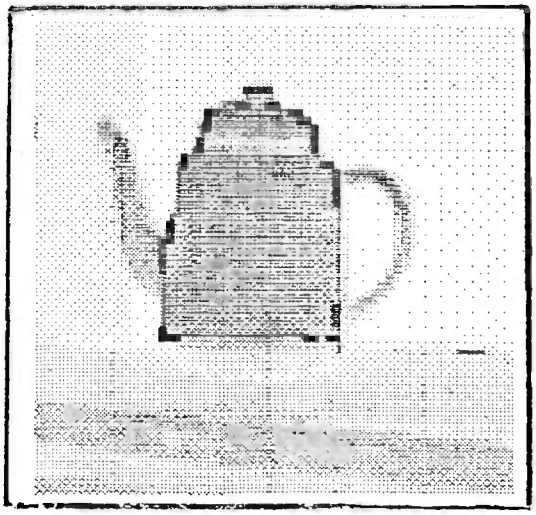
e)



f)

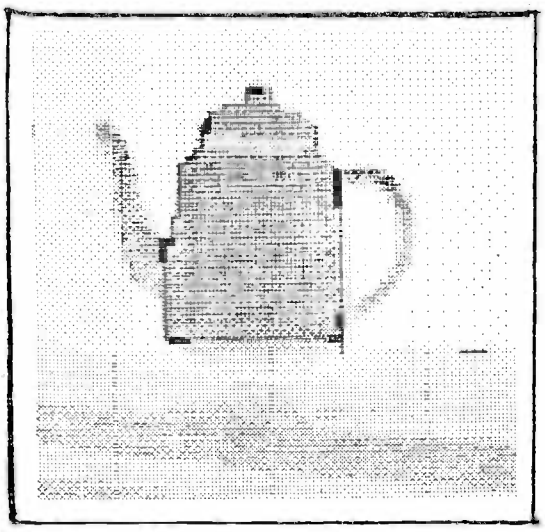


g)

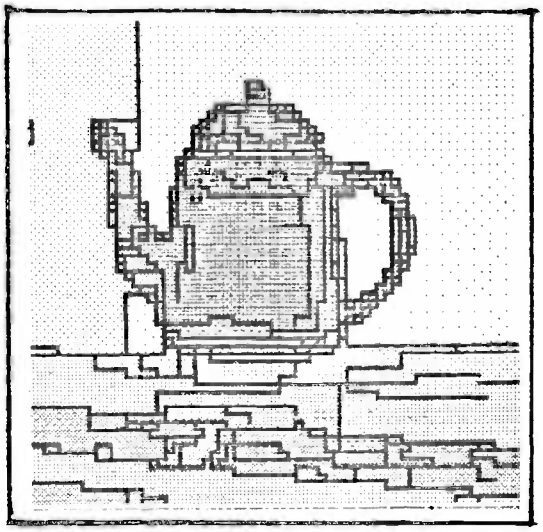


h)

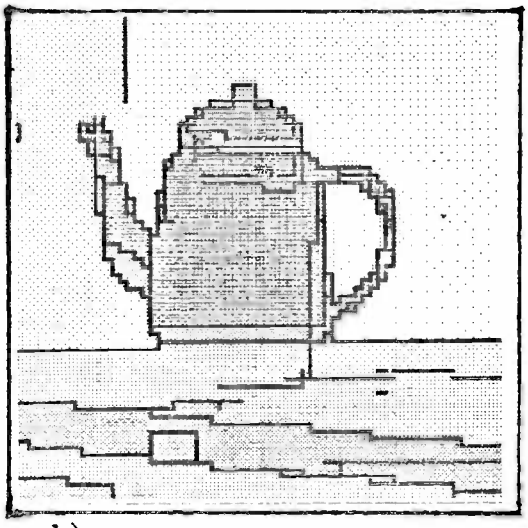
Fig F.1 cont.



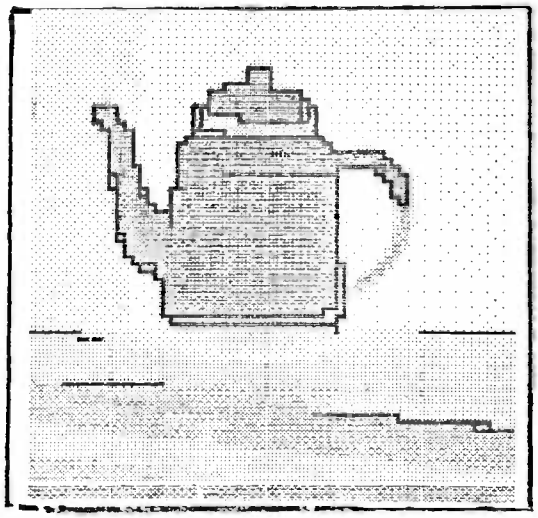
i)



j)

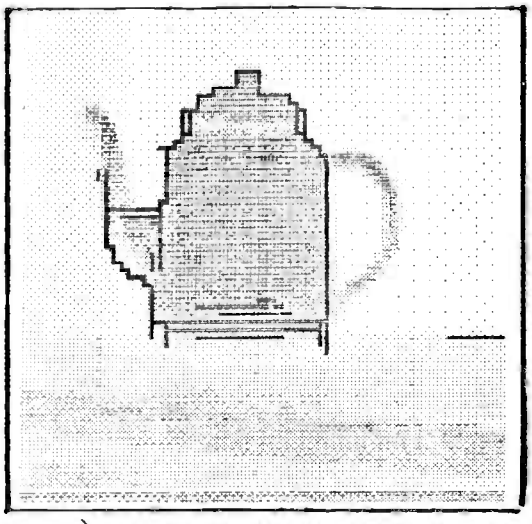


k)



1)

Fig F.1 cont.



m)

Fig F.1 cont.

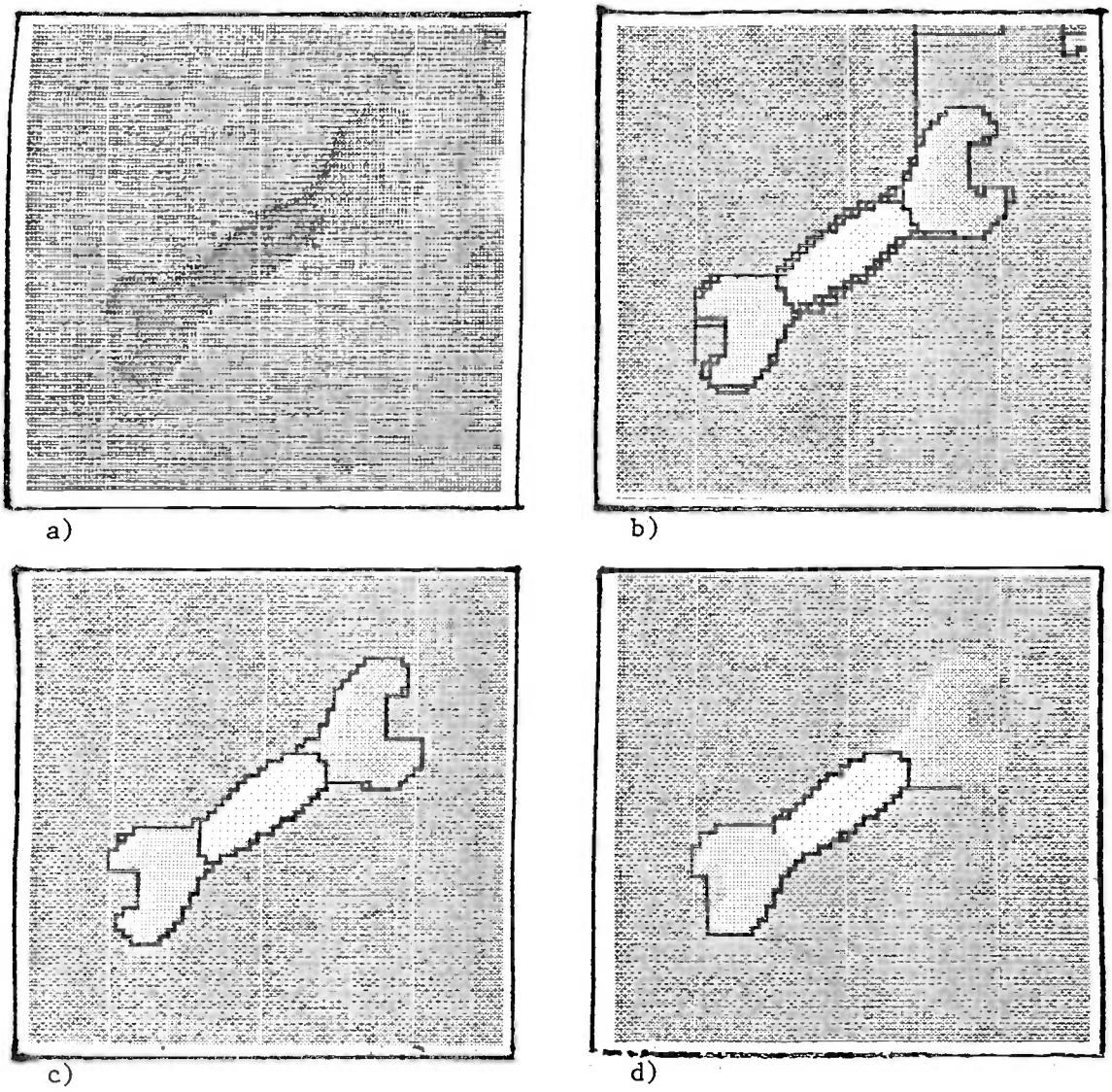
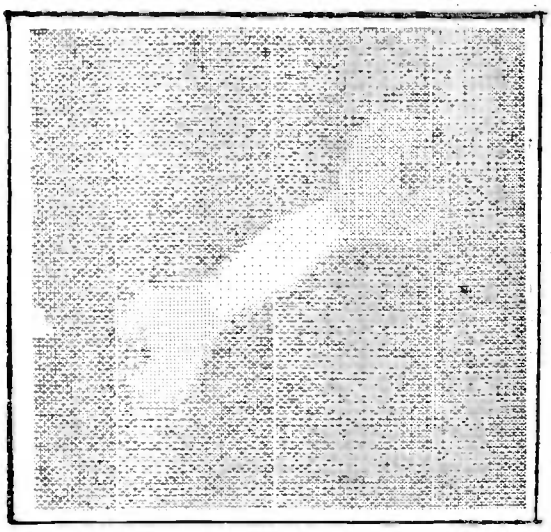
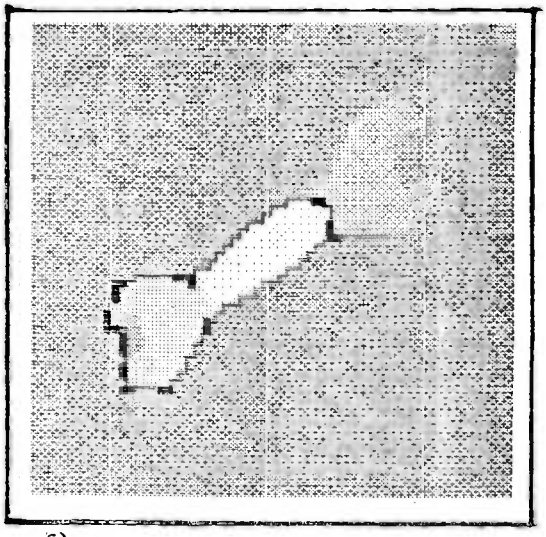


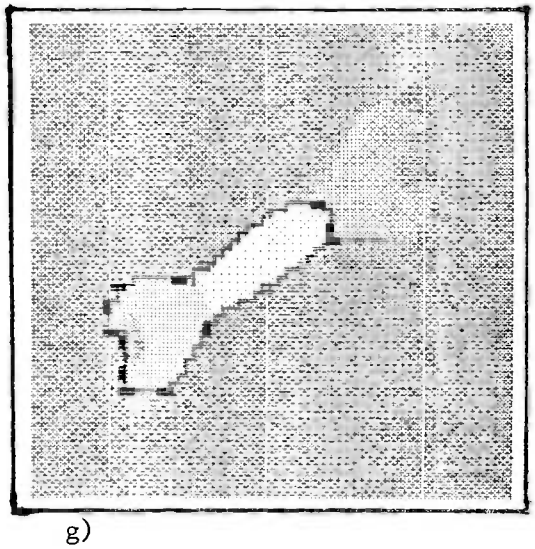
Fig F.2 Weak constancy relaxation on image (a) to define intensity discontinuities (b-e), using penalty C=32,64,128,256 respectively. Weak constancy angle relaxation: (f&g) from the lines in (d) with C=50,100 respectively. Intensity relaxation using the "economy regime" (h-k), C=32,64,128,256.



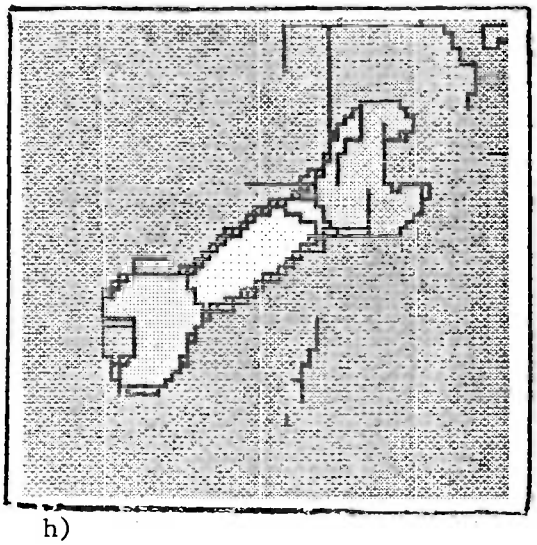
e)



f)

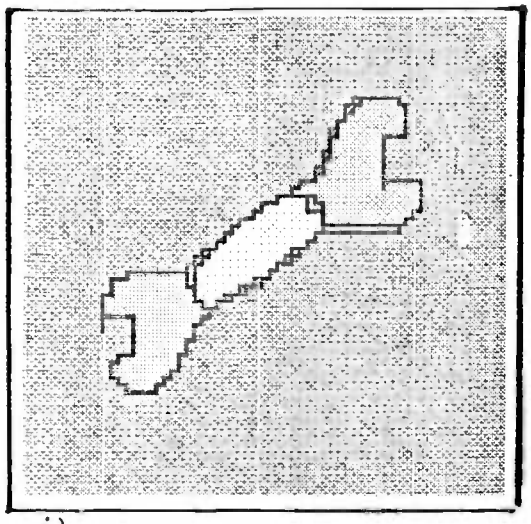


g)

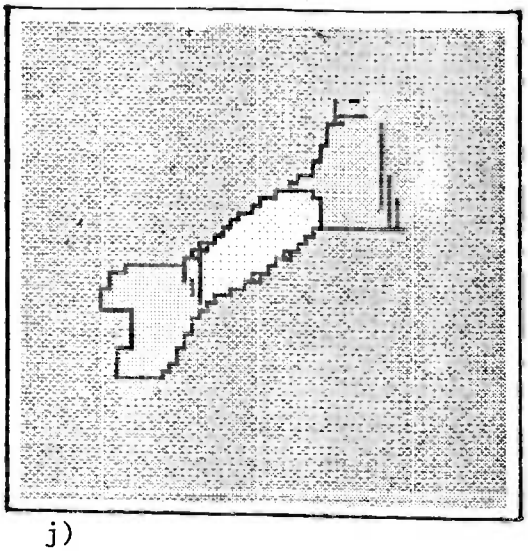


h)

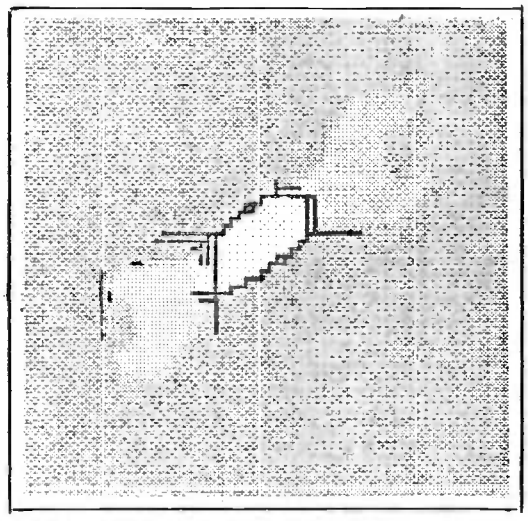
Fig F.2 cont.



i)



j)



k)

Fig F.2 cont.

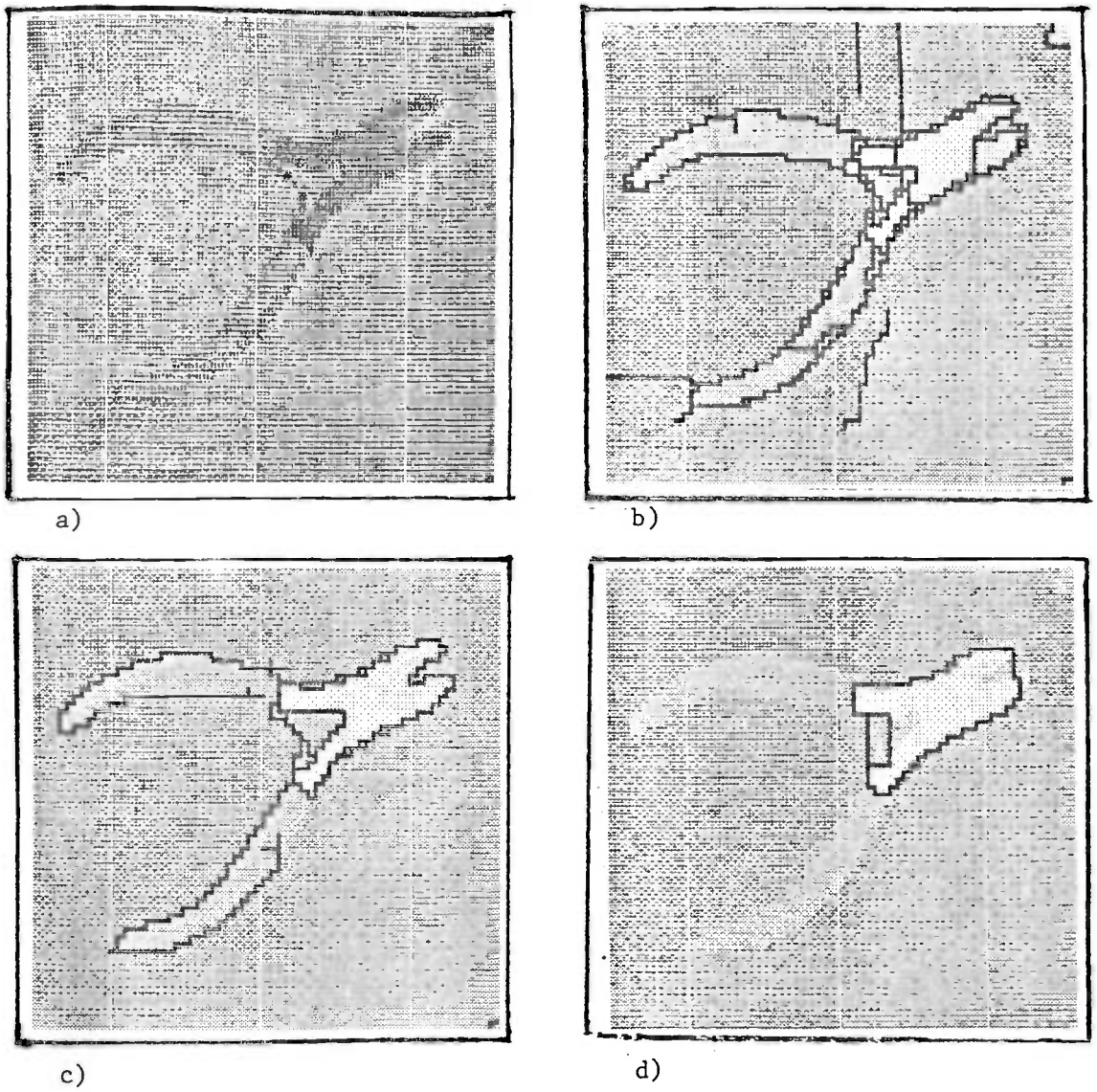
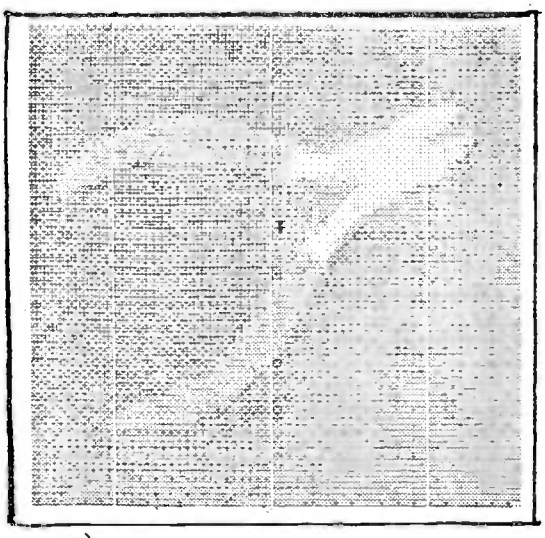
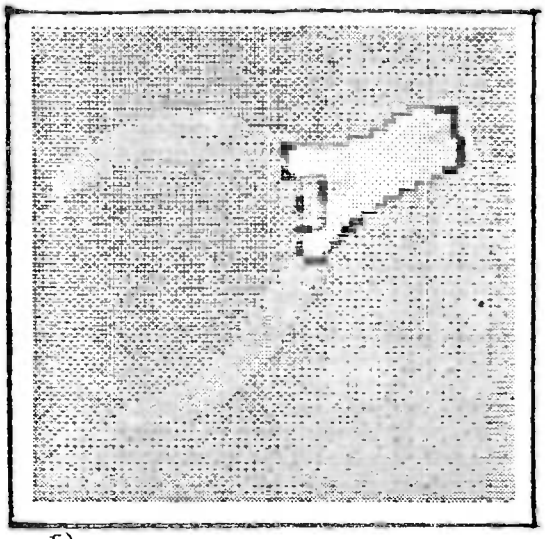


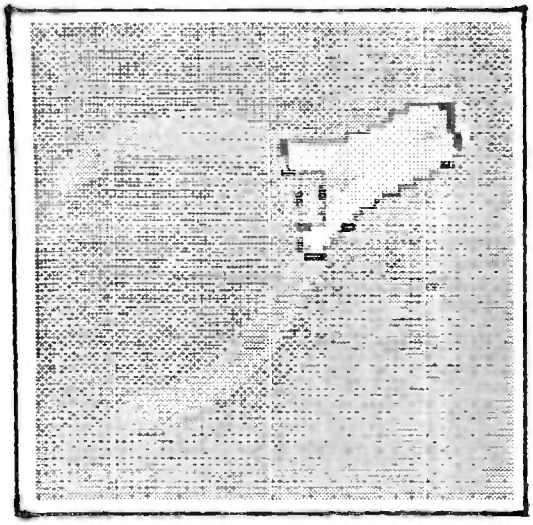
Fig F.3 Weak constancy relaxation on image (a) to define intensity discontinuities (b-e), using penalty C=32,64,128,256 respectively. Weak constancy angle relaxation: (f&g) from lines in (d) with C=50,100 respectively. Intensity relaxation using the "economy regime" (h-k), C=32,64,128,256.



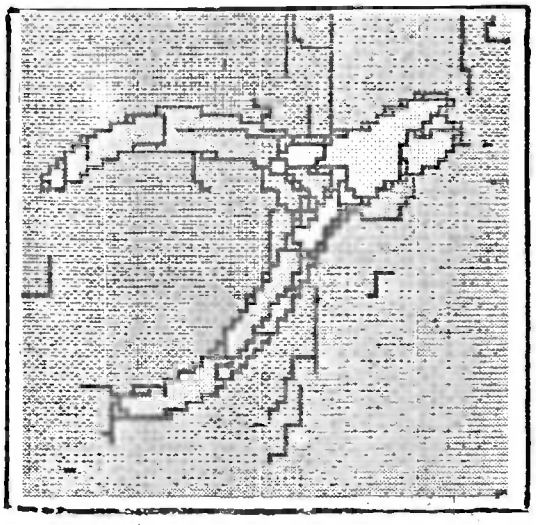
e)



f)

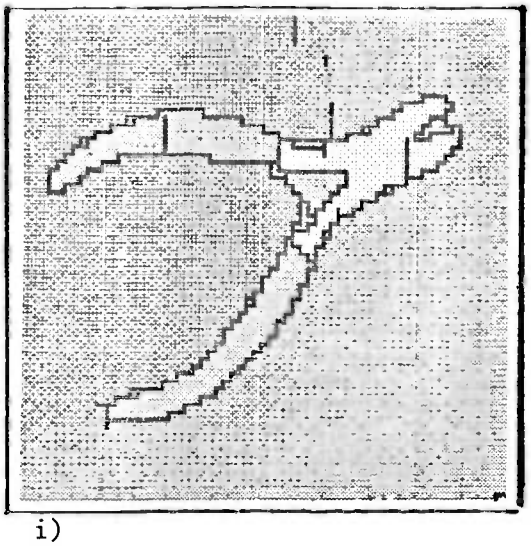


g)

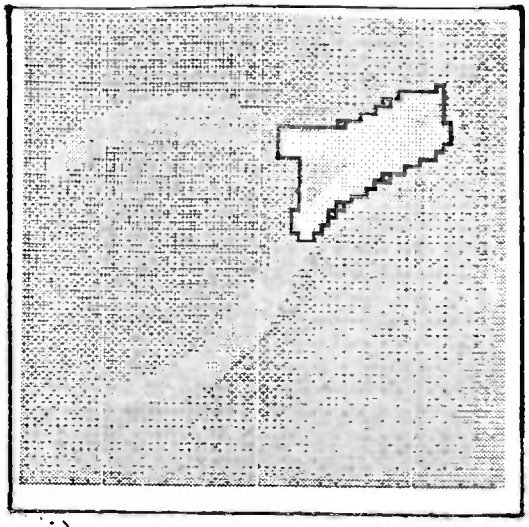


h)

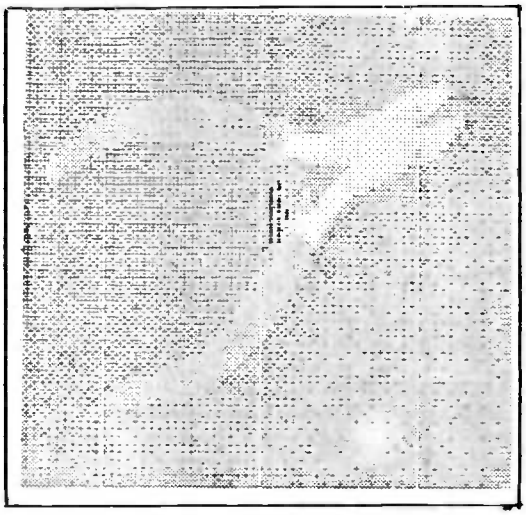
Fig F.3 cont.



i)



j)



k)

Fig F.3 cont.

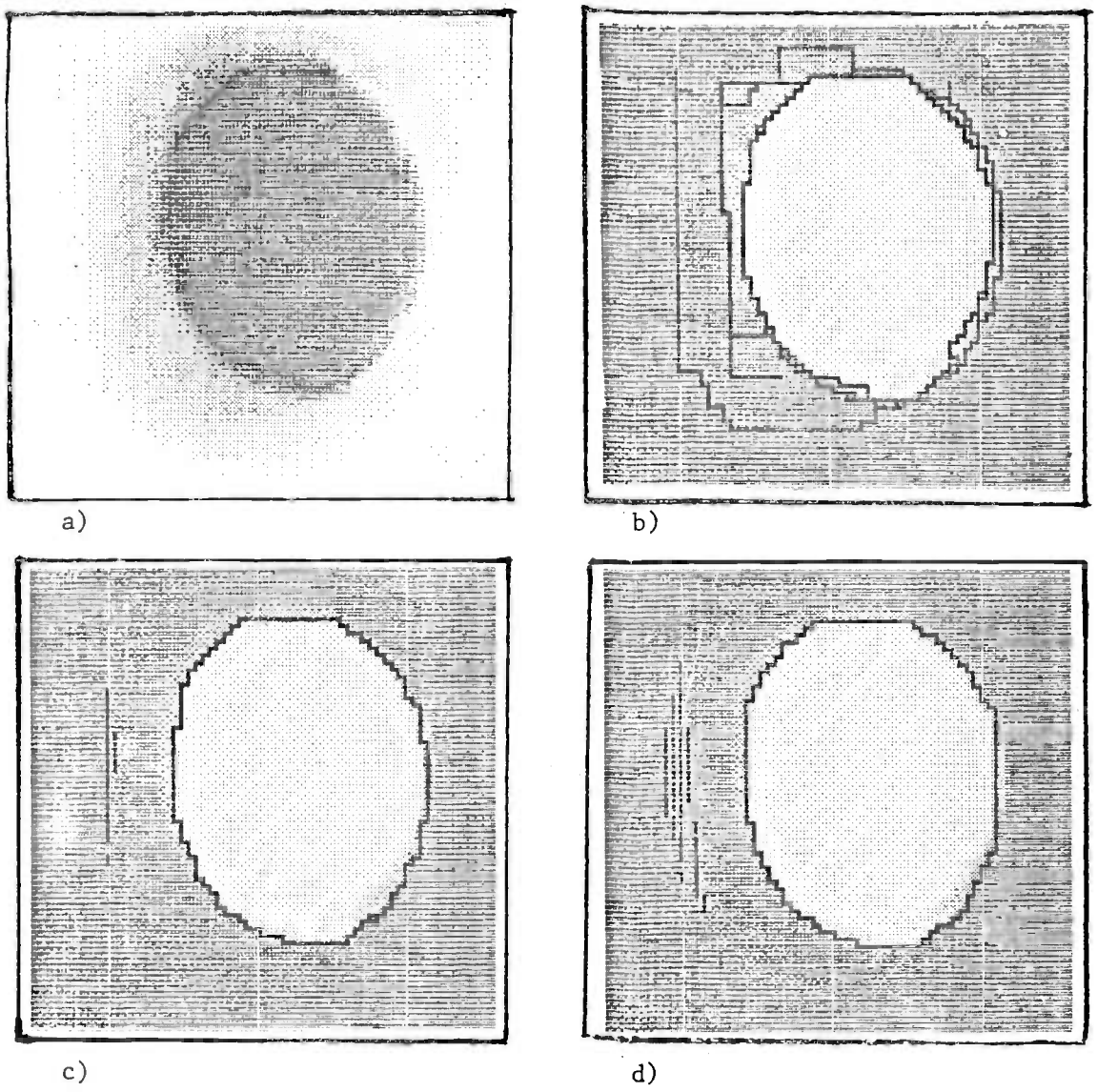


Fig F.4 Weak constancy relaxation on image (a) to define intensity discontinuities (b-d) using penalty C=64,128,256 respectively.

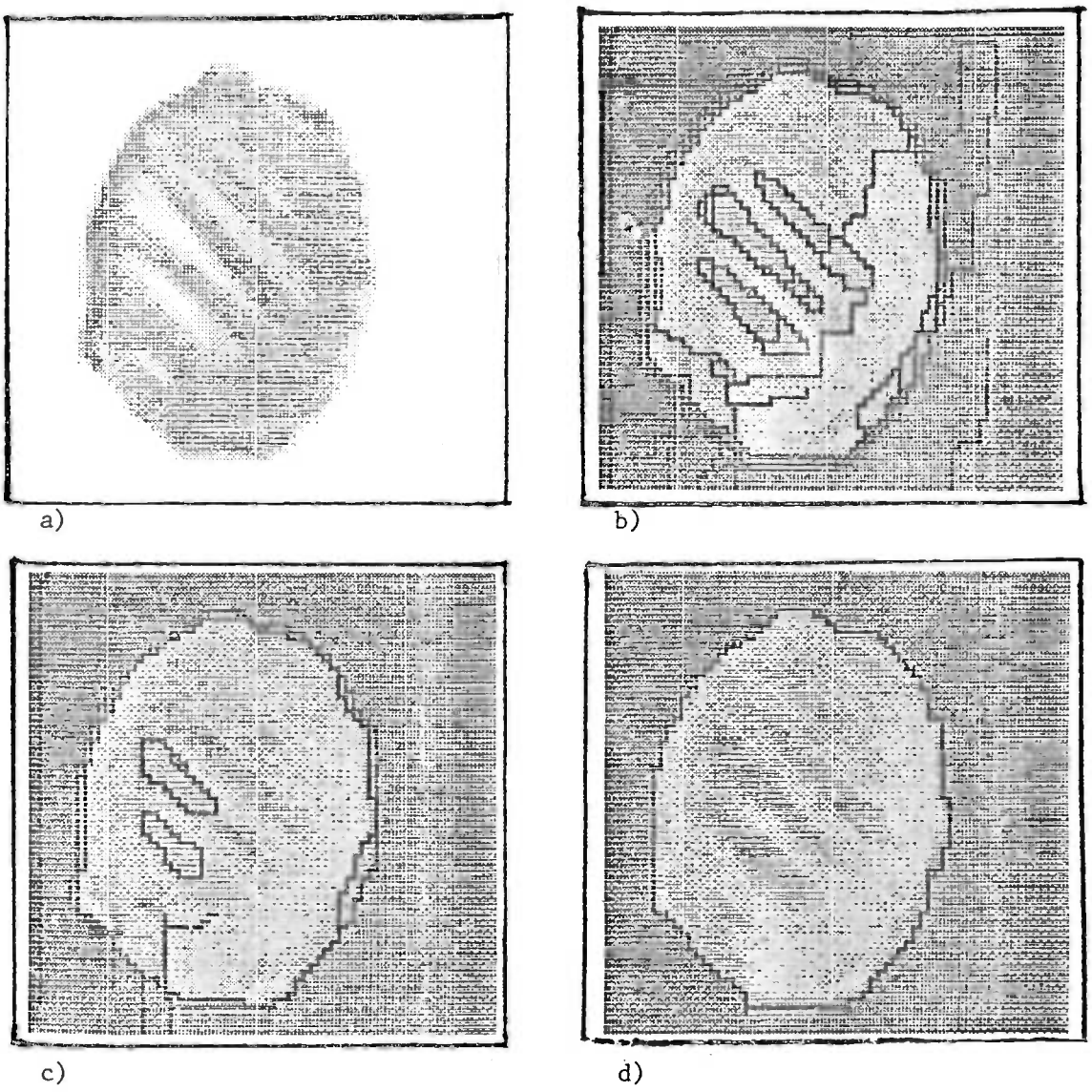


Fig F.5 Weak constancy relaxation on image (a) to define intensity discontinuities (b-d) using penalty C=32,64,128 respectively.

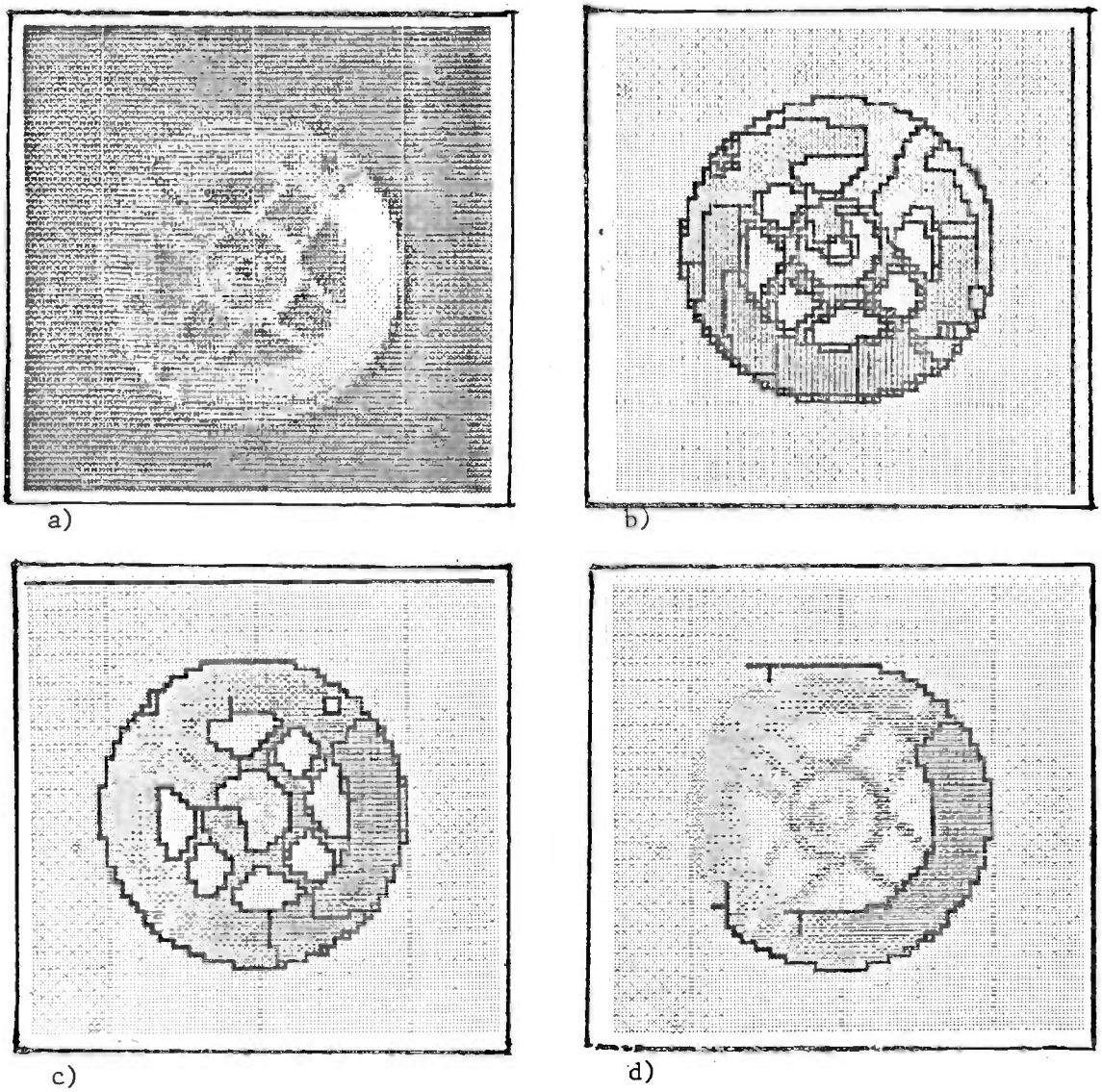


Fig F.6 Weak constancy relaxation on image (a) to define intensity discontinuities (b-d) using penalty C=32,64,128 respectively.

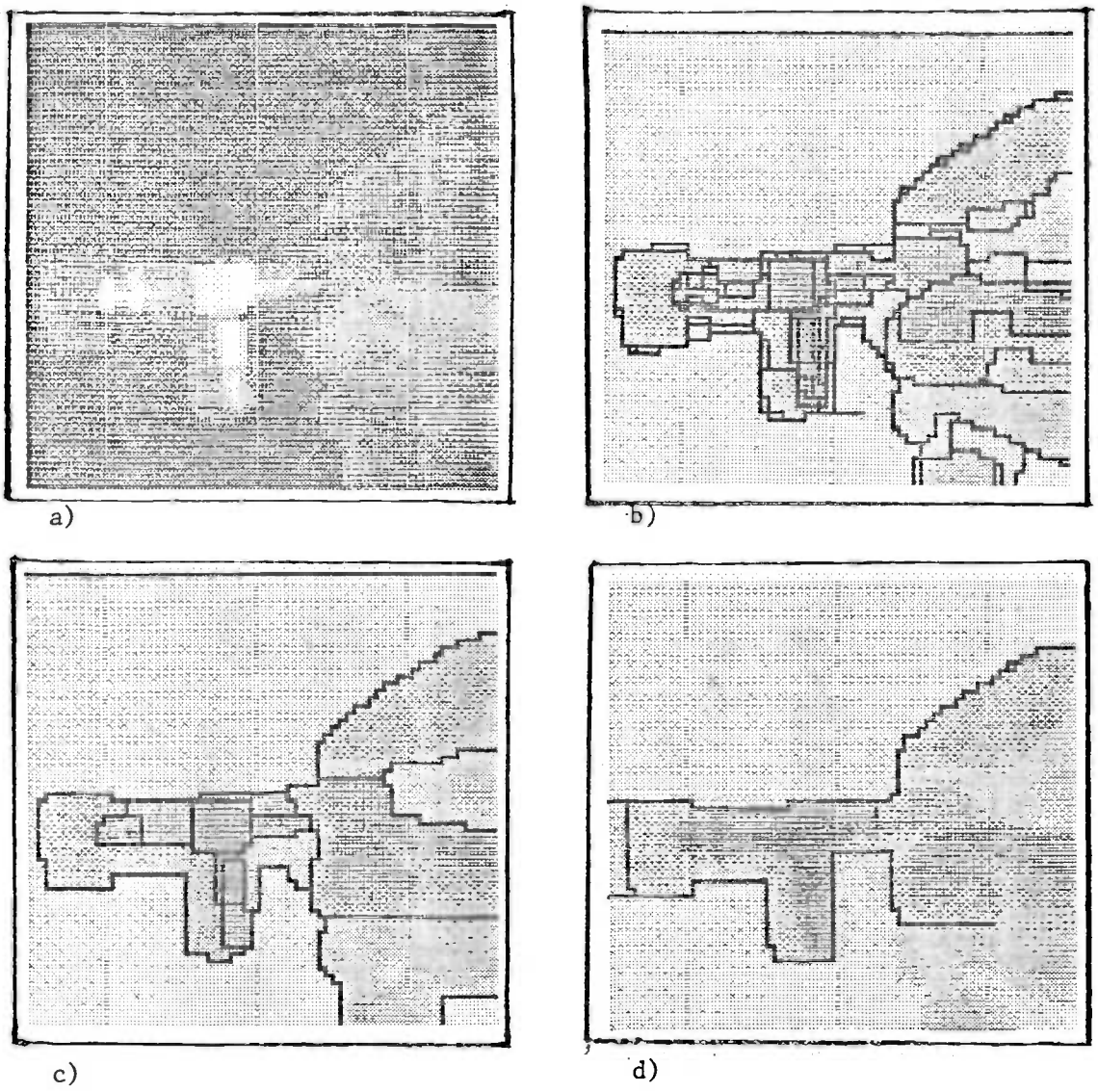


Fig F.7 Weak constancy relaxation on image (a) to define intensity discontinuities (b-d) using penalty C=32,64,128 respectively.

G Programs in PARAPIC and C for relaxation

```
! PARAPIC Program to impose weak constancy constraints on an array !


!Globals !
vars  x,x0,     !current and original pics - double precision !
      mask,   !chequer board mask !
      costplus, costminus, ! cumulated adjustment costs
                  for increment/decrement !

      c1,cc,c3,c4,c5,c6,f, ! constants, set in function iterate !
      inc,          ! current relaxation increment !
      F,d,              ! points of discontinuity in the relaxation fn !

function getcost;
vars dir, dirs;     !direction of current neighbour pixel !

 ! images. !
vars  cplus,   ! (change in cost) for a change of +inc in x !
      cminus,     ! -(change in cost) for a change -inc in x !
      u,    ! edge strength !
      s,    ! scratch plane !
      mp1, mp2, mm1, mm2; ! masks corresponding to pieces of cost fn !


      undef->costplus;    ! total cplus !
      undef->costminus, ! total cminus !
      initc(1)->dirs;       !dir as a string !

      1->dir;
      until dir>2 then
          x-x|>dir -> u;

          dir+ #0 -> subscrc(1,dirs),
          frame(dirs)->s;

          ((u<F)&(u>=(-F))) &~s -> mp1;
          ((u<d)&(u>=(-d))) &~s -> mp2;

          ((u=<F)&(u>(-F))) &~s -> mm1;
          ((u=<d)&(u>(-d))) &~s -> mm2;

          ! 0 -logshift(u,c3)->s;  BUT c3=0, so instead: !
          0-u -> s;
          mp1?(s,0) -> cplus;
          mm1?(s,0) -> cminus;

          logshift(u.sign,c4) -> s;
          (mp1&~mp2)?(cplus+s,cplus) -> cplus;
          (mm1&~mm2)?(cminus+s,cminus) -> cminus;

          logshift(u,c5) -> s;
          mp2?(cplus+s+c6,cplus) -> cplus;
          mm2?(cminus+s-c6,cminus) -> cminus;

          if (dir=1) then
              cplus -> costplus;
```

```
            cminus -> costminus;
        else
            cplus + costplus -> costplus;
            cminus + costminus -> costminus;
        close;

    ! shift to add in cost from the opposite neighbour (symmetry) !
        costplus - cminus|>(dir+2) -> costplus;
        costminus - cplus|>(dir+2) -> costminus;

        dir+1->dir;
    close;

    logshift(x-x0,c1) -> s; ! add in quadratic cost of displacement !
    costplus + s -> costplus;
    costminus + s -> costminus,

end;

 ! call getcost and update x !

function update => change;
vars neg;

    .getcost;
    costplus < (-cc) -> costplus,
    costminus > (cc) -> costminus,
    costminus &~ costplus -> costminus, ! do one or t'other, not both !

    (mask&costplus)?(x+inc,x) -> x;
    (mask&costminus)?(x-inc,x) -> x;

    x<0->neg;
    neg?(0,x) -> x;       !keep x positive !
    ~ mask -> mask;   ! toggle chequerboard mask !

    area((costplus|costminus)&~neg) -> change;
            !report no of alterable pixels !
end;


 ! set up consts and iteratively update !

function iterate c,p,i, N;  !must have i>= -3 and c>=2+i  !

 ! c,p,i, are cost, penalty (>=0), increment (=<0), given as logs to
base 2. N is max iter !

vars n,change;

!rescaling !
    if i>0 then
        0->i;
    close,
    if  (12-i>15) then
        errfun('','danger of overflow');
```

```
        close;

        if (p+c+3-i-15 -> ov; ov<0) then
            0->ov;
        close;

        i+ov->i;

        logshift(x0,-i) -> x0;
        logshift(x,-i) -> x;
        c-i->c;

        c+p->f,
        2^p->p;
        p-sqrt(p^2 -1) -> p;
        p*(2^c) -> F;
        intof(F+0.5) -> F;
        4 -> d,

        3 -> c1;
        2 -> cc; != 4*c2 !
        0->c3;
        f -> c4;
        f-2 -> c5;
        logshift(1,f-3) -> c6;

        1 -> inc;

 !we now have
represented as logs:
        c1 c3 c4 c5 f
represented as numbers:
        cc c6 inc F d
 !

 !set up chequerboard mask !
        logand(1.ramp,1) = logand(2.ramp,1) -> mask;

        1->n;
        1->change;
        pr('(iteration,change): ');
        while (n=<N) and (change/=0) then
            .update -> change ;
            '('.pr;n.pr;',';.pr; change.pr,')'.pr; 1.sp;
            n+1->n;
        close;
        1.nl;

        !reset scale of x,x0 !
        logshift(x0,i)->x0,
        logshift(x,i)->x;
end;


function relax p c =>p;
        logand(8:377,p.extend)->x; !unsigned !
```

```
        x->x0;

        iterate(c,0,0,20);
        iterate(c,0,-3,30);

        iterate(c,1,0,20);
        iterate(c,1,-2,20);

        iterate(c,2,-1,20);
        iterate(c,2,-3,30);

        iterate(c,3,-3,40);

        x.lower->p;
end;
```

```
/* Program in C to impose weak constancy constraint on an image array

to run, type:
        wconst [-c#][-p#.#][-d#][-i#][-r]  <picfile> > outfile
*/


#include <stdio.h>
#include <math.h>


short tabspace[4103] {};
short *table = tabspace+2051;/* lookup table */
short nbits = 0;        /* picture scaling: nbits +8 bits */
short A,B,C,D,E,mE,E_2,F,mF,F1,mF1,f,d {};/* global constants */
short x[4096], x0[4096] {};/* adjusted, initial data */
char apad1[64],active[4096],apad2[64];/* activity flags */
short asize, area, asize_2 {},/* pic dimensions */

char *progname;
char rflag = 0,
short ival = -4;
short dval = 16;
float pval = 0.5;

FILE *fp = NULL;      /* input picture channel */

short change = 0,     /* no of altered pixels in a cycle of adjust */

float costq(), costp(), truecost();



main(argc,argv)
char **argv,
{
short coarg;


      progname = *argv++;

      while ((argc>1)&&((*argv)[0] == '-'))
      {
          switch((*argv)[1]) {

          case 'r':     /* report costs */
              rflag = 1;   /* report costs */
              argv++;
              break;

          case 'c':     /* penalty for breaking a constraint */
              sscanf(*argv++,"-c%d",&coarg);
              break;

          case 'i':     /* set precision */
```

```
            sscanf(*argv++,"-i%d",&ival);
            ival = -ival;
            break;

        case 'd':    /* set smoothness constant */
            sscanf(*argv++,"-d%d",&dval),
            break,

        case 'p':    /* set function index ratio */
            sscanf(*argv++,"-p%f",&pval);
            break,
        default:
            error("unknown flag");
            break,
        }
        --argc;
    }

    if (argc>1)
    {
        if((fp = fopen(*argv++,"r")) == NULL)
            error("can't access picture file");
        --argc;
    }
    else
        fp=stdin;

    readpic();
    fclose(fp);

    relax(coarg);

    scale(0);
    writepic();
}



relaxp(c,p,inc,imax,cmax)
float p;
{
short iter;


    if ((iter = iterate(c,p,inc,imax,cmax)) == -1)
        fprintf(stderr,"bad ");
    else
    {
        fprintf(stderr,"%d ",iter);
    }
    return(iter);
}



relax(c)
```

```
{
short i, iter,  itersum, il;
float p,pl,cst,icst,cst1;

        itersum=0;
        iter=0,
        il=4;
        fprintf(stderr,"c=%d0,c);
        if (rflag)
        {
            scale(-ival);
            icst= costq() + truecost(c);
            fprintf(stderr,"initial cost=%.3f0,icst);
        }

        for (i=0,i>= ival; --i)
        {
            if ((iter= relaxp(c,1.0,i,100,0))  != -1)
            {
                itersum+=iter;
                il=i;
            }
        }

        itersum += relaxp(c,1.0,il,200,0);
        if (rflag)
        {
            cst=costq(),
            fprintf(stderr,"0rue cost = %.3f. ",cst+truecost(c) );
            fprintf(stderr,"convex cost = %.3f. ",(cst+=costp()) );
        }
        fprintf(stderr,"0recision= %d0,il);

        for (p=pval, p>0.05; p*= pval)
        {
            if (rflag)
                fprintf(stderr,"p=%.4f  ",p);
            for (i=0; i>il; --i)
            {
                if((iter=relaxp(c,p,i,100,0)) != -1)
                    itersum+=iter;
            }
            if((iter=relaxp(c,p,il,200,0)) != -1)
            {
                itersum+=iter;
                pl=p;
                if (rflag)
                    fprintf(stderr,"true cost = %.3f0,
                     (cst1=costq()+truecost(c)) );

            }
        }

        if (rflag)
            fprintf(stderr,"subopt = %.2f   ",cst1-cst);
        fprintf(stderr,"8d iterations0, itersum),
```

```
}


error(s)
char *s;
{
        fprintf(stderr,"%s: %s0,progname,s);
        exit(1);
}



readpic()/*read from fp, set up x,x0 as integer arrays*/
{

register short *p, *q;
short i, *pend;


        asize = getc(fp);
        if ((asize<8)||(asize>64))
            error("array size out of range");
        area = asize*asize;
        asize_2 = asize<<1;

        if (getc(fp)  != asize)
            error("picture size error");

        nbits = getc(fp)-8;

        for (i=509; i; --i)
            getc(fp);    /* read rest of block -MIRU format */

        for (p=x,q=x0,pend=x+area; p<pend;)
        {
            *p = 0377& getc(fp);
            *q++ = *p++;
        }
}



writepic()     /* send x out on stdout. low bytes only */
{

register short *p;
short *pend,i;

        putchar(asize);
        putchar(asize);
        putchar(' 10');
        for (i=509; i; --i)
            putchar(' ');
```

```
        for (p=x, pend=x+area; p<pend;)
            putchar(*p++);
}


setuptab()    /* fill lookup-table entries */
{

register short *p,u;
short k;


        for (u= -d,p=table-d; u<d, u++)
            *p++ = A + u*B - u*D;

        k= -1+C;
        for (u=d, p=table+d, u<F; u++)
            *p++ = k -(u*D);

        k= -1-C;
        for (u= -F, p=table-F; u< -d; u++)
            *p++ = k - (u*D);

        p=table+F;
        *p= *(p+1) = *(p+2) =0;
        p=table-F-1;
        *p = *(p-1) = *(p-2) = 0;

}


scale(n)  /* scale x,x0 to n+8 bits */
{

short s, *ptop,
register short *p,*q;


        s=n-nbits;
        nbits=n;

        if (s>0)
        {
            for (p=x,q=x0,ptop=x+area; p<ptop;)
            {
                *p++ <<= s,
                *q++ <<= s;
            }
        }
        else if (s<0)
        {
            s= -s;
            for (p=x,q=x0,ptop=x+area, p<ptop;)
            {
                *p++ >>= s;
                *q++ >>= s,
```

```
/* macros for adjust */

#define qcost     {psum= *p- *q; psum <<=E_2; nsum=psum;}

#define dircost(n)    {u= *p- *(n); if((u>= -F)&&(u<=F)) {
                psum+= table[u]; nsum += table[u-1];} }

#define north     dircost(p-asize)
#define south     dircost(p+asize)
#define east dircost(p+1)
#define west dircost(p-1)

#define report    {change++; *actp= *(actp+1)= *(actp-1) = *(actp+asize)
          = *(actp-asize) =1,}

#define alter{if (psum< mE) { (*p)++ ; report ;} else if ((nsum>E)
          &&(*p>0)) { --(*p) ; report;} else *actp=0;}

#define loop(s,e,a)   for (actp=active+(s), atop=active+(e);
               actp<atop, actp+=a)  if (*actp)

#define setp {i=actp-active, p=x+i; q=x0+i;}



adjust()  /* make one iteration of adjustments to
          all array elements. special conditions
          at borders. */
{
      long psum, nsum;
      short *p, u, i;
      short *ptop, offset, row, last, *q, off;
      register char *actp, *atop;


      change =0;
      off=asize+1,

      /* interior of array */
      for (row=1, last=asize-1; row<last; row++)
      {
          for (actp=active+off, atop=actp+asize-2; actp<atop; actp++)
              if (*actp)
              {
                  setp;
                  qcost;
                  north, east; south; west;
                  alter;
              }
          off+=asize,
```

```
    }


/* North row */
    loop(1,asize-1,1)
    {
        setp;
        qcost;
        east; south; west;
        alter;
    }

/* South row */
    loop(area-asize+1,area-1,1)
    {
        setp;
        qcost;
        east; north; west;
        alter;
    }

/* West column */
    loop(asize,area-asize,asize)
    {
        setp;
        qcost;
        east; north; south;
        alter;
    }

/* East column */
    loop(2*asize-1,area-1,asize)
    {
        setp;
        qcost;
        west; north; south;
        alter;
    }


/* NW corner */
    p=x;q=x0;actp=active;
    qcost;
    south;east;
    alter;

/* NE corner */
    p=x+asize-1;q=x0+asize-1;actp=active+asize-1;
    qcost;
    south;west;
    alter;

/* SW corner */
    p=x+area-asize;q=x0+area-asize;actp=active+area-asize;
    qcost,
    north;east;
```

```
        alter;

/* SE corner */
        p=x+area-1;q=x0+area-1;actp=active+area-1;
        qcost;
        north;west;
        alter;

        return(change);
}


iterate(cost,penalty,inc,maxiter,maxchange)
float penalty;
{

float freal, Freal;
short iter;
char *actp;

        for (actp=active+area, actp>active;)
            *--actp =1;      /* initially all active */
        if (inc< -4) /* increment is 2^ -inc */
            return(-1);

        scale(-inc);
        cost = cost<<(-inc);
        Freal= cost*penalty;
        if ((penalty< 0.01)||(penalty> 1.0))
            return(-1);
        freal= 0.5*(penalty+ (1/penalty))* ((float) cost);

        if (freal>16300.0)
            return(-1);
        f= (short) (freal);
        F = (short) (Freal );
        if (F>2048)
            return(-1);
        F1=F+1;
        mF= -F;
        mF1= -F1;

        d=dval;
        while(d>F/4)
            d>>=1;
        A = (f/d) -1;
        C = f<<1;
        B = C/d;
        D = 2;
        E = 8;
        mE = -E;
        E_2 = 4;     /* E*2 as a log */

        setuptab();

        change=maxchange+1,
```

```
        iter=0;
        while ((change>maxchange) && (iter<maxiter))
        {
            adjust();
            iter++;
        }
        return(iter);
}




float costq()  /* quadratic component of cost */
{
long ctot,c;
short *p, *q, *ptop, nb;
float retval;

        ctot=0;
        nb= -nbits;
        for(p=x,q=x0,ptop=x+area;p<ptop;)
        {
            c= (*p++ - *q++)<<nb;
            ctot+=c*c;
        }

        retval=ctot;
        retval /= area;
        retval*=E;
        return(retval);
}


float truecost(c) /* constraint breaking penalties */
{
long cd, ctot;
short i,k, *p, *ptop, lF,d;
float retval,

        ctot =0;
        cd = c*c;   /* penalty c squared */

        lF = (c<<(-ival))/20;/* 1/20 = 0.05, last penalty value */
        d=dval;
        while (d>lF/4)
            d>>=1;

        for(i=1; i<asize; i++)
            for(p=x+i*asize, ptop=p+asize; p<ptop; p++)
            {
                k= (*p- *(p-asize));
                if ((k>d)||(k< -d))
                    ctot += cd;
            }
```

```
        for(i=0; i<asize;i++)
            for(p=x+i*asize, ptop=p+asize-1; p<ptop; p++)
            {
                k= (*p- *(p+1));
                if ((k>d)||(k< -d))
                    ctot += cd;
            }

        retval=ctot;
        retval/=area;
        return(retval);
}



float costp() /* neighbour interaction component of cost */
{
long cd, cc, cf, ci, ctot;
short i,k, ns, *p, *ptop;
float retval,

        ctot =0;
        cc = C;
        cf = F*(cc-F);
        cd = F*cc;
        ns = -2*nbits;

        for(i=1, i<asize; i++)
            for(p=x+i*asize, ptop=p+asize; p<ptop; p++)
            {
                k= (*p- *(p-asize));
                if (k<0)
                    k= -k;
                if (k>F)
                    ci=cf;
                else
                    ci= k*(cc-k);
                ctot += ci << ns;
            }


        for(i=0; i<asize,i++)
            for(p=x+i*asize, ptop=p+asize-1; p<ptop; p++)
            {
                k= (*p- *(p+1));
                if (k<0)
                    k= -k;
                if (k>F)
                    ci=cf;
                else
                    ci= k*(cc-k);
                ctot += ci << ns;
            }

        retval=ctot;
        retval/=area;
```

```
        return(retval);
}
```

```
 ! PARAPIC program for relaxation with edge-region and minimum
strength constraints. Using double precision integers.  !



vars INC; 2->INC; !iteration rate constant. rate 2^-INC. INC>=0  !
vars MINU; 1->MINU; !precision required before termination. ok cos
                u.mod.greymax is mon. decr. !



 ! cut off below minimum strength and prevent overflow !
function stretch edge t1 => edge;
vars s;
       edge<0 ->s;
       edge.mod-> edge;
       (edge=<t1)?(0,edge)->edge;
       (edge>63)?(63,edge) -> edge;
       s?(0-edge,edge)->edge;
       edge.extend->edge;
       logshift(edge,6)->edge;
end;


function display e => e;  !rescale edges for display !
       logshift(e,-4)->e;
       e.mod->e;
       (e>255)?(255,e.lower)->e;
end;


 !boolean valued sign function !
function sgn p => b;
       p<0 -> b;
end;


function curl h v =>s;
       h - v ->s;
       s - h|>2 -> s;
       s + v|>1 ->s;
end;


function report i s u v;
      i.pr; 1.sp; s.mod greymax.pr; 1.sp;
      u.mod greymax.pr;1.sp; v.area.pr;
      1.nl;
end;


function relax(pic, t1, maxiter) => hor ver;
vars hmask, vmask, esgn, hstripes, vstripes, violated; !boolean planes !
vars s,u;  !double planes !
```

```
vars i,j;       !iteration counter !

      stretch(pic - pic|>1,t1)->hor; !rescale edges !
      stretch(pic - pic|>2,t1)->ver;
      0->pic;  !save space !


      (hor/=0)  ->hmask, !protect edges for min
                 strength constraint !
      (ver/=0)  ->vmask;

! masks to split each of hor,ver into 2 non-interacting sets !
      logand(2.ramp,1)=0 -> hstripes,
      logand(1.ramp,1)=0 -> vstripes,

! initialise constraint array, s, and iteration counter, i. !
      curl(hor,ver) -> s;
      0->i;
      1.initb->violated;
      report(i,s,s,violated);

! relaxation loop !
      until (violated.nowhere or (i>=maxiter)) then
        0->j;
        0.initb->violated;
        while(j+1->j,j=<2) then

      !horizontal edges !
         logshift(s-s|>4, -(1+INC)) -> u; !calculate edge strength
                          alteration !
          !deal with overflow and mask for currently updatable edges !
         hor.sgn->esgn;
         (((u.sgn==esgn)|(esgn==sgn(hor-u)))&hmask&hstripes)?(u,0) -> u;
         violated|(u mod>MINU) -> violated;
                  ! record where there is no updating !

         hor - u -> hor;  ! update edge strengths !
         s - u +u|>2 ->s; ! update values of constraints !
         ~ hstripes -> hstripes; ! mask alternates each iteration !

      !vertical edges !
         logshift(s-s|>3, -(1+INC)) -> u;

         ver.sgn->esgn;
         (((u.sgn@esgn)|(sgn(ver+u)==esgn))&vmask&vstripes)?(u,0) -> u;
         violated|(u.mod>MINU) -> violated;
         ver + u -> ver;
         s - u + u|>1 -> s;
         ~ vstripes -> vstripes;

        close;
        i+1 -> i;
        report(i,s,u,violated);
      close;
end;
```